

Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

Tímový projekt

# Softvérovo riadené siete rozšírené o WiFi štandard

Projektová dokumentácia - riadenie

Vedúci projektu: Ing. Rastislav Bencel  
Názov tímu: MGWA - Make Wifi Great Again  
Členovia tímu: Bc. Katarina Bedejová  
Bc. Matej Belluš  
Bc. Marcel Sabol  
Bc. Michal Gottstein  
Bc. Marián Hesek  
Bc. Michal Hampel  
Bc. Miroslav Procházka  
Pracovisko: FIIT STU - 5.45  
Akademický rok: 2017/2018  
Dátum odovzdania: 11.5.2018

# Obsah

<b>Obsah</b>	<b>2</b>
<b>1 Úvod</b>	<b>6</b>
<b>2 Členovia tímu a ich roly</b>	<b>7</b>
2.1 Predstavenie členov tímu	7
Ing. Rastislav Bencel (Product owner)	7
Bc. Katarina Bedejová (Human Markets Coordinator)	7
Bc. Matej Belluš (Dynamic Implementation Director)	7
Bc. Marcel Sabol (Global Factors Strategist)	7
Bc. Michal Gottstein (Dynamic Creative Developer)	7
Bc. Marián Hesek (Central Security Designer)	7
Bc. Michal Hampel (Corporate Tactics Director)	7
Bc. Miroslav Procházka (Human Quality Producer)	7
2.2 Rozdelenie manažérskych úloh	8
<b>3 Aplikácie manažmentov</b>	<b>9</b>
<b>4 Sumarizácie šprintov</b>	<b>11</b>
4.1 Šprint 1 - "testovací"(1-týždňový)	11
4.2 Šprint 2 - (2-týždňový)	11
4.3 Šprint 3 - (2-týždňový)	11
<b>5 Používané metodiky</b>	<b>12</b>
5.1 Metodika dokumentácie	12
5.2 Metodika písania kódu	12
5.3 Metodika komunikácie	12
5.4 Metodika verziovania a prehliadok kódu	12
5.5 Metodika testovania	12
5.6 Metodika úloh	13
<b>Úvod</b>	<b>15</b>
802.11 generátor	16
LVAP	16
Use case - prijatie 802.11 manažment rámca	16
Use case - prijatie 802.11 dátového rámca	16
<b>OpenFlow</b>	<b>17</b>
Úvod	17
Fungovanie	17
Možnosti	17

<b>Floodlight</b>	<b>18</b>
Úvod	18
Možnosti	18
Rozšírenia	18
Generovanie BSSID	18
<b>Open vSwitch</b>	<b>20</b>
Úvod	20
Možnosti	20
<b>OpenFlow</b>	<b>21</b>
<b>Prílohy</b>	<b>22</b>
A Metodiky	22
<b>Úvod</b>	<b>25</b>
<b>Metodika generovania dokumentácie</b>	<b>26</b>
Floodlight	26
Základné dokumentačné tagy	26
Opis dokumentačných tagov s príkladmi ich použitia	26
OpenvSwitch	29
<b>Metodika tvorby dokumentácie</b>	<b>31</b>
<b>Obsah</b>	<b>33</b>
<b>1. Úvod</b>	<b>34</b>
<b>2. Metodika pomenovaní (naming conventions)</b>	<b>35</b>
Java	35
C	36
<b>3. Metodika komentárov</b>	<b>37</b>
Java	37
C	37
<b>Obsah</b>	<b>39</b>
<b>Úvod</b>	<b>40</b>
<b>Osobné stretnutia</b>	<b>41</b>
<b>Elektronické komunikačné kanály</b>	<b>42</b>
Obsah	44
<b>Úvod</b>	<b>45</b>
<b>Slovník pojmov - Github</b>	<b>45</b>
<b>Základný workflow</b>	<b>45</b>

<b>Code review</b>	<b>45</b>
<b>Obsah Github Wiki</b>	<b>46</b>
Consult some git tutorials:	46
Create new branch:	46
Example commit:	46
Example Push:	46
rollback a github repository to a specific commit	46
Obsah	48
<b>1 Úvod</b>	<b>49</b>
<b>2 Floodlight - Unit Tests</b>	<b>49</b>
2.1 Písanie nového testu	50
2.2 Príklad JUnit Testu	50
<b>3 Open vSwitch - Unit Tests</b>	<b>51</b>
Obsah	53
<b>1 Úvod</b>	<b>54</b>
<b>2 Spravovanie úloh v ZenHube</b>	<b>55</b>
2.1 Vytvorenie úlohy	55
2.2 Vymazanie úlohy	58
2.3 Pridelenie a estimovanie úlohy	58
2.3 Stavy úloh	58

# Dokumentácia k riadeniu

# 1 Úvod

Dokument vznikol v rámci predmetu Tímový projekt 1 a Tímového projektu 2. Dokumentuje prácu na projekte Softvérovo riadené siete rozšírené o WiFi štandard. Obsahom tejto dokumentácie je opis manažmentu v tíme. Špecifikácia vytváraného systému je bližšie opísaná v dokumentácii inžinierskeho diela.

Dokument obsahuje informácie o tíme a vývoji projektu. Nachádza sa tu opis jednotlivých členov tímu, ich prínos pre projekt a ich skúsenosti. Ďalej sa v dokumente nachádzajú informácie o manažmente projektu - o priebehu stretnutí, výstupoch stretnutí a používaných nástrojoch, či už komunikačných, alebo vývojových.

Obsahom tejto časti dokumentu sú aj informácie o jednotlivých šprintoch - ich trvanie, problémy riešené v rámci šprintu, výstupy a pod. Jedna kapitola je venovaná metodikám, ktoré pri práci na projekte dodržiavame. Či už sú to nízkoúrovňové metodiky (týkajúce sa programovania a využívania nástrojov), alebo vysokoúrovňové metodiky. Na záver sa v dokumente nachádzajú exporty z nástroja evidencie úloh.

## 2 Členovia tímu a ich roly

### 2.1 Predstavenie členov tímu

Ing. Rastislav Bencel (Product owner)

Študent doktorantského štúdia na FIIT, dlhoročné skúsenosti s problematikou SDN a WiFi.

Bc. Katarina Bedejová (Human Markets Coordinator)

Absolvent bakalárskeho štúdia na FIIT odbor Informatika. V bakalárskej práci sa venovala vývoju súborového systému v jazyku C sprístupňujúcemu obsah dostupný cez sieť.

Bc. Matej Belluš (Dynamic Implementation Director)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. V bakalárskej práci sa venoval problematike optimalizácii WiFi v súvislosti s SDN. Má rád linux a open source projekty. Vo svojej kariere už pracoval ako programátor a aj ako linux server admin. Na FIIT STU študuje, aby si zdokonalil vedomosti o sieťach a bezpečnosti.

Bc. Marcel Sabol (Global Factors Strategist)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. Rád skúša nové veci a nebojí sa výziev. Momentálne pracuje v známej firme Accenture ako Java programátor.

Bc. Michal Gottstein (Dynamic Creative Developer)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. V rámci bakalárskej práce sa venoval IoT, konkrétne návrhu a implementácii parkovacích senzorov za pomoci siete Sifgox. Počas štúdia pracuje ako frontend-developer, vo voľnom čase sa venuje grafike a dizajnu.

Michal po prvom semestri ukončil inžinierske štúdium. Jeho úlohy sme si rozdelili podľa potreby medzi zvyškom tímu.

Bc. Marián Hesek (Central Security Designer)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. Popri škole pracuje ako tester rôznych aplikácií. Jeho skúsenosti v oblasti testovania nám pomôžu zabezpečiť dodávku funkčného kódu.

Bc. Michal Hampel (Corporate Tactics Director)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. Skúsenosti, ktoré nabral na pozícii Java programátora nám určite pomôžu v riešení skrytých problémov.

## Bc. Miroslav Procházka (Human Quality Producer)

Absolvent bakalárskeho štúdia na FIIT odbor Internetové technológie. Odborník na virtualizáciu a prepojenie hardverových zariadení.

### 2.2 Rozdelenie manažérskych úloh

Člen tímu	Pozícia v tíme
Matej Belluš	Manažment plánovania a verziovania
Katarina Bedejová	Manažment dokumentácie
Michal Hampel	Manažment kvality
Marian Heseck	Manažment testovania
Miroslav Procházka	Manažment zberu požiadaviek
Marcel Sabol	Manažment úloh
Michal Gottstein	Manažment komunikácie



## 3 Aplikácie manažmentov

### Manažment dokumentácie

V rámci manažmentu dokumentácie vytvárame priebežné a jednotlivé dokumenty. Do priebežných dokumentov zaraďujeme zápisnicu z každého tímového stretnutia, retrospektívu po každom šprinte a dokumentáciu zdrojového kódu. Tieto dokumenty sa buď priebežne menia, alebo sa priebežne vytvárajú nové verzie podľa potreby. Do jednotlivých dokumentov zaraďujeme tímové metodiky, dokumentáciu riadenia a dokumentáciu inžinierskeho diela. Každý z vytváraných dokumentov v rámci tímového projektu má zadanú šablónu, podľa ktorej sa tento dokument vytvára.

### Manažment komunikácie

Komunikáciu môžeme rozdeliť na komunikáciu na stretnutí a komunikáciu mimo stretnutia. V rámci komunikácie na stretnutí má každý člen tímu priestor povedať o svojich výsledkoch a problémoch, ktoré nastali, a ak je to nutné, otvoríme priestor pre hlbšiu diskusiu o vzniknutých problémoch. Komunikácia mimo stretnutia funguje hlavne v nástroji Discord. Pre rôzne účely sme vytvorili viacero kanálov podľa druhu témy, ktorej sa daná komunikácia týka. Ďalej je dôležité mať zavedené efektívne fungovanie komunikácie s produktovým vlastníkom, ktorým je zároveň aj náš vedúci tímu. Na túto komunikáciu používame taktiež nástroj Discord. Vedúceho taktiež informujeme o aktuálnom stave projektu a úloh na pravidelných týždenných tímových stretnutiach.

### Manažment zberu požiadaviek

V rámci manažmentu zberu požiadaviek priebežne vytvárame požiadavky pre náš projekt spolu s produktovým vlastníkom. Zber požiadaviek prebieha väčšinou počas priamej komunikácie s produktovým vlastníkom. Pokiaľ sú navrhnuté nejaké požiadavky, musia byť dôkladne analyzované. Až po analyzovaní sa môže rozhodnúť, či budú požiadavky akceptované v tej forme, aká bola navrhnutá. Ak nastane prípad, že požiadavky z istého dôvodu nemôžu byť naplnené, je nutné ich pozmeniť, respektíve od nich úplne upustiť.

### Manažment plánovania

#### Plánovanie v rámci tímu

Na začiatku semestra sme vytvorili základné používateľské príbehy produktu aj s popisom, teda minimálne požiadavky, ktoré by mali byť splnené. Na stretnutí pred začiatkom šprintu vždy podľa priority používateľských príbehov zoberieme ich časť, pre ktorú sa zaviazame, že ju stihneme spraviť. Tieto používateľské príbehy následne rozdelíme na menšie úlohy, ktoré obsahujú všetko to, čo je potrebné splniť pre úspešné vykonanie používateľského príbehu. Jednotlivé úlohy sa pridelia členom tímu tak, aby každý člen tímu mal určenú robotu na šprint a taktiež tak, aby sme vedeli, že má predstavu o tom, čo bude robiť. Po začatí šprintu je o týždeň stretnutie v rámci šprintu, kde rozoberáme všetko, čo sa podarilo za uplynulý týždeň členom tímu vytvoriť/vyriešiť a taktiež ťažkosti, s ktorými sa stretli alebo museli

vysporiadať a navrhujeme možné riešenia. Ak je to potrebné, vytvoríme ďalšie úlohy, ktoré pri prvotnom plánovaní nebolo zrejmé, že treba vyriešiť.

#### Stav projektu

Stav projektu je reflektovaný v stavoch úloh, ktoré sú v produktovom backlogu. Tieto stavy úloh sú pravidelne aktualizované. Všetky úlohy, používateľské príbehy, epické príbehy, ako aj záznamy o chybách môžeme nájsť v nástroji ZenHub.

#### Stav úloh

Ku každej úlohe je uvedený jej názov, popis (description), priorita úlohy a odhad (v počte story pointov), koľko bude táto úloha trvať. Pred tým, ako sa začne robiť na úlohe, musí mať úloha jasne určené, kto na nej bude pracovať.

#### Zhodnotenie šprintu

Po každom šprinte na stretnutí tím prezentuje produktovému vlastníkovi nový prírastok k produktu a aj všetky splnené a nesplnené úlohy.

### **Manažment testovania**

V rámci testovania môžeme testovanie rozdeliť na dve časti - konkrétne na testovanie kódu a testovanie vizuálneho výstupu. Testovanie kódu je zabezpečované samotným vývojárom, ktorý daný kód implementoval. Toto testovanie bude prebiehať pomocou unit testov implementovaných v kóde.

### **Manažment úloh**

V rámci manažmentu úloh je vytvorená metodika manažmentu úloh, ktorá opisuje postupy na správu úloh v nástroji ZenHub. Je potrebné mať určené ako vytvárať, priradovať a dokončovať úlohy. Úlohy sa vytvárajú na základe konzultácii pred začiatkom šprintu. Metódou "planning poker" určujeme náročnosť úloh. Úlohy sú priradované členom tímu počas plánovania šprintu a vykonávajú sa podľa priority. Nižšie číslo určuje väčšiu prioritu. Manažér úloh je zodpovedný za správu všetkých úloh v projekte v nástroji ZenHub.

### **Manažment kvality**

Na zabezpečenie dostatočnej kvality jednotlivých úloh máme zadefinované rôzne metodiky, ako napr. metodiku prehliadky kódu a verziovania, metodiku písania kódu a taktiež tímový "definition of done".

## 4 Sumarizácie šprintov

V tejto kapitole sú sumarizované zatiaľ absolvované šprinty. Šprint je obdobie, za ktoré vyvojový tím dodá malú funkčnú časť. V našom prípade sme do šprintu zaraďovali aj úlohy na analyzovanie technológií a úlohy na vytváranie dokumentácie.

### 4.1 Šprint 1 - “testovací”(1-týždňový)

V prvom testovacom šprinte bolo našou úlohou zoznámiť sa s témou a vytvoriť úvodné kroky k tomu, aby sme vedeli dobre spolupracovať. Nakoľko téma nadväzuje na dizertačnú prácu nášho product ownera Rast'a, bolo potrebné aby nám tému objasnil a načrtnol smerovanie našej práce na tomto projekte. K prvotným inicializačným úlohám patrilo vytvorenie plagátu, návrh loga a založenie webového sídla. Následne bolo potrebné aby sme si našťudovali viacero odborných článkov, ktoré súviseli s našou problematikou a pomohli nám pochopiť technológie, s ktorými budeme pracovať. Vytvorili sme a nasadili webovú stránku tímového projektu. Začalo sa pracovať na vytváraní jednotlivých metodík.

Počas retrospektívy sme zhodnotili celý šprint jednotlivo, ako aj tímovo. Každý člen tímu zhodnotil priebeh šprintu zo svojho pohľadu. Keďže išlo o testovací šprint a úlohy boli jednoduché, podarilo sa nám ich splniť všetky. Každý navrhol, ako by bolo možné nasledujúci šprint vylepšiť.

Na záver sme estimovali a zaradili úlohy do ďalšieho šprintu.

### 4.2 Šprint 2 (2-týždňový)

V druhom šprinte sme sa zamerali na implementačné úlohy. Počas celého šprintu sme pracovali na svojich taskoch, bohužiaľ, pre zlý odhad náročnosti úloh a chýbajúcej dokumentácii existujúceho projektu, bol tento šprint vo veľkej miere neúspešný.

V retrospektíve sme prediskutovali šprint. Rozpoznali sme klady a zápory a navrhli sme zlepšenia do ďalšieho šprintu. Taktiež sme identifikovali veci, ktorým sa chceme v budúcnosti vyvarovať. Tak isto sme vyjadrili pochybenie nad kompatibilitou nášho zadania a SCRUM metodiky.

### 4.3 Šprint 3 (2-týždňový)

Naše obavy z predchádzajúceho šprintu o tom, že metódy SCRUM nie sú pre náš projekt úplne ideálne, sme konzultovali na predmete Manažment v tvorbe softvéru s doc. Ing. Marián Šimko, PhD. Ten nám prisľúbil, že sa stretne s vedúcim nášho projektu s Ing. Rastislavom Bencelom, o možnostiach riešenia nášho problému.

Počas šprintu sme zaznamenali prvé pokroky pri implementácii úloh. Podarilo sa nám implementovať jednoduchšie úlohy ako napr. SSID generátora, na druhej strane komplexnejšie úlohy vyžadujúce viac analýzy ostali opäť neuzatvorené. Počas sprintu bolo taktiež potrebné vymeniť si vedomosti z naštudovanej problematiky, preto niektorí členovia si pripravili prezentácie, ktoré pomohli k pochopeniu problematiky celému tímu.

Retrospektíva nám ukázala, že naše časové ohodnotenie úloh nie je správne. Nakoľko je potrebné zanalyzovať veľké množstvo kódu, nestíhame implementačné úlohy.

## 4.4 Šprint 4 (3-týždňový)

Stretnutie doc. Ing. Marián Šimko, PhD. a Ing. Rastislava Bencela sa do dnešného dňa neuskutočnilo.

Počas šprintu sme sa venovali problematike OVS simulátora, posielaním bacon rámcov cez wifi a snažili sme sa pokročiť s ovládaním Wifi ovládača.

V retrospektíve sme sa zhodli na tom, že je potrebné vytvárať oveľa viac dokumentácie k analyzovanému kódu. Vytvárať akúsi mapu, ktorá nám poslúži na lepšiu orientáciu v kóde. Taktiež sme navrhli novú metódu riešenia problémových taskov, ktorá spočíva v 20 minútovej analýze kódu každého člena tímu. Na konci tejto individuálnej analýzy je spoločné zhodnotenie jej výsledkou. Túto metódu sme si hneď aj vyskúšali a zdá sa byť veľmi účinná.

## 4.4 Ďalšie Šprinty

Ďalšie zápisnice zo šprintov sú zverejnené na stránke tímového projektu v časti venovanej dokumentácií.

# 5 Používané metodiky

## 5.1 Metodika dokumentácie

V rámci metodiky vytvárania dokumentácie existujú pravidlá, ktoré definujú, ako dokumentáciu vytvárať a taktiež rozsah dokumentácie. Mimo tímovej dokumentácie riadenia a dokumentácie vytváraného inžinierskeho diela sa pravidelne na každom stretnutí tímu vytvárajú zápisnice stretnutí a po ukončení šprintu sa vytvárajú retrospektívy daného šprintu. Čo sa týka dokumentácie zdrojového kódu, tá je generovaná pomocou nástrojov a postupov definovaných v metodike

## 5.2 Metodika písania kódu

V rámci metodiky písania kódu, sme si zadefinovali formát ako budeme písať názvy premenných a metód/funkcií v príslušnom programovacom jazyku. Jazyk C a Java sú dominantné programovacie jazyky, ktoré budeme v našom tímovom projekte najviac

využívať a preto je táto metodika zadefinovaná pre tieto jazyky. Takisto sme nezabudli na zadefinovanie pravidiel pre písanie komentárov.

## 5.3 Metodika komunikácie

V rámci metodiky komunikácie, sme zadefinovali spôsoby osobnej a elektronickej komunikácie v našom tíme. Hlavným elektronickým komunikačným prostriedkom je ZenHub a nástroj Discord, v ktorom sú vytvorené rôzne komunikačné kanály. Metodika komunikácie zahrňuje prípady pre použitie rôznych druhov elektronických komunikačných kanálov. V tejto metodike sú zahrnuté aj komunikačné kanály ako: Facebook, telefón a email.

## 5.4 Metodika verziovania a prehliadok kódu

V rámci metodiky verziovania je zadefinované ako a kedy sa vytvárajú pull requesty, commity a kedy sa spájajú dve vetvy kódu. Ďalej je tu detailne opísané ako sa robí prehliadka kódu, s čím súvisia aj pravidlá ktoré musí kód spínať, aby bol akceptovaný.

## 5.5 Metodika testovania

V rámci metodiky testovania sme si zaviedli spôsob testovania technológií, ktoré využívame v našom projekte. Uvideli sme si ako spúšťať unit testy používaných technológií a podľa akých konvencií vytvárať nové testy.

## 5.6 Metodika úloh

Táto metodika ma poskytuje návod na spravu úloh v rámci projektu. Opisuje aká hierarchia úloh sa v projekte používa. Ako jednotlivé typy úloh existujú a akú abstrakciu predstavujú. Ďalej sú tu opísané stavy, ktoré môže úloha nadobúdať. Na spravu úloh je použitý nástroj ZenHub (nadstavba nad GitHubom). Metodika úloh je určená všetkým členom tímu. Jej pochopenie a aplikácia je dôležitá pre trvalo udržateľný vývoj projektu.

# Dokumentácia inžinierskeho diela

Vedúci projektu: Ing. Rastislav Bencel  
Názov tímu: MGWA - Make Wifi Great Again  
Členovia tímu: Bc. Katarina Bedejová  
                  Bc. Matej Belluš  
                  Bc. Marcel Sabol  
                  Bc. Michal Gottstein  
                  Bc. Marián HeseK  
                  Bc. Michal Hampel  
                  Bc. Miroslav Procházka  
Pracovisko: FIIT STU - 5.45  
Akademický rok: 2017/2018  
Dátum odovzdania: 11.5.2018

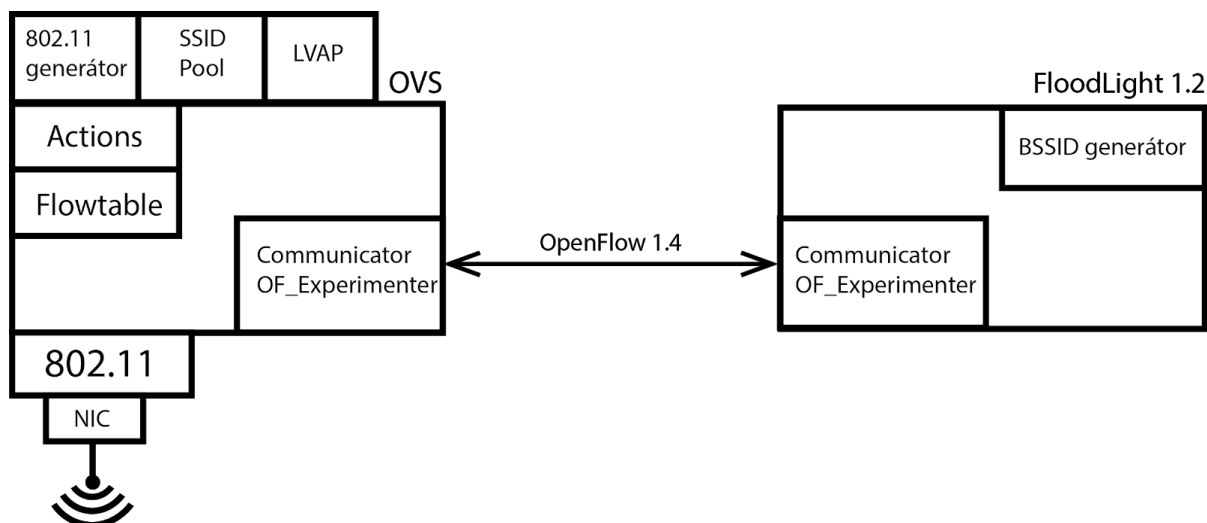
# Úvod

Tento dokument obsahuje informácie o stave projektu počas prvých 3 šprintov. Hlavným cieľom tohto dokumentu je oboznámenie čitateľa s technickou stránkou projektu. Dokument obsahuje podrobný opis kľúčových technológií a štandardov.

Softvérovo riadené siete (SDN – Software-Defined Networking) prinášajú centralizovaný pohľad na počítačovú sieť oproti tradičnému distribuovanému prístupu, čo pre bezdrôtové siete predstavuje ich lepšie manažovanie a optimalizovanie využívania ich prostriedkov. SDN však v súčasnosti neobsahujú priamu podporu bezdrôtových sietí. V súčasnosti existuje viacero bezdrôtových sietí (3G, LTE, rôzne senzorové siete, Bluetooth, WiFi ....), no v rámci projektu sa zameriame na WiFi štandard a to pre podnikové (WiFi enterprise) siete, ktoré sú charakteristické veľkým počtom uzlov. V súčasnosti existujú riešenia, ktoré prinášajú podporu WiFi štandardu. Tieto riešenia vytvárajú nové riadiace protokoly už k existujúcim riadiacim protokolom (napr. OpenFlow), čo nepredstavuje čisté architektonické riešenie. Cieľom projektu je prinesenie WiFi štandardu do SDN, so zameraním sa na zachovanie konceptu SDN a čistotu riešenia po architektonickej stránke. Projekt pozostáva v návrhu architektúry vychádzajúcej z princípu SDN (definovanie WiFi funkcionality pre jednotlivé uzly siete) a v následnej modifikácii existujúceho protokolu OpenFlow pre zabezpečenie funkcionality architektúry. Okrem vytvorenia dobre popísanej architektúry je cieľom aj jej implementácia, ktorá bude tvoriť odrazový mostík pre ďalšiu výskumnú činnosť.

Základom tohto projektu je rozšíriť existujúce softvérovo riadené siete SDN o WiFi štandard. Pre dané rozšírenie je potrebné vytvoriť návrh architektúry podľa konceptu SDN, v ktorom je potrebné definovať funkcionality jednotlivých komponentov a tiež rozšírenie OpenFlow protokolu, aby spĺňal existujúci špecifikáciu. Okrem vytvorenia návrhu je cieľom aj implementácia architektúry, ktorá tiež slúži ako jej overenie.

Cieľom projektu je vytvoriť ilúziu samostatného AP pre každého klienta pripojeného k jednému fyzickému AP s použitím technológie SDN. Projekt sa skladá z dvoch hlavných častí: Open vSwitch a SDN controller Floodlight. Obidve sú už plne funkčné samostatné aplikácie. Našou úlohou je modifikovať tieto aplikácie tak, aby spĺňali požiadavky.



Obr. 1. Kompletné riešenie sa skladá z viacerých častí (modulov).

## Rozhranie 802.11

Komunikáciu medzi 802.11 stanicami a naším riešením obsluhuje rozhranie 802.11. Toto rozhranie obsluhuje sieťové rozhranie a komunikáciu medzi ním, okolitým svetom a komponentami softvérovo riadených sietí. Úlohou tohto rozhrania je prijímať rámce z bezdrôtovej WiFi siete, odpovedať na ne alebo ich preposielať do OpenVSwitch, a prijímať správy z OpenVSwitch a preposielať ich do siete WiFi.

Kedže samotný OVS nevie, čo má s prijatými 802.11 rámcami robiť, je potrebné túto funkcionálnu doimplementovať. Tento modul má na starosti vytáčať korektné 802.11 rámce so správnymi poliami. Pod generovaním sa myslí generovanie napr.: *Probe* odpovede alebo *Beacon* správ v pravidelnom časovom intervale.

### Funkcionálne požiadavky

Rozhranie 802.11 musí spĺňať nasledovné funkcionálne požiadavky:

- Bezdrôtová komunikácia medzi rozhraním a stanicami
- Komunikácia medzi rozhraním a OpenVSwitch
- Vysielanie *Beacon* rámcov
- Odpovedanie na *Probe* požiadavky
- Autentifikácia klienta
- Asociácia klienta
- Uchovávanie *LVAP* informácií

### Implementácia

Začiatok pokusov implementácie generovania správ protokolu 802.11 bola knižnica *libtins*, ktorá je implementovaná v jazyku C++. Komplexnosť tejto knižnice však nespĺňala naše požiadavky na implementáciu, čiže komplexnosť implementácie generovania rámcov



protokolu 802.11 nebola úmerná k požiadavkám. Toto nás prinútilo hľadať iné riešenie. Siahli sme preto po knižnici *scapy*, ktorá je implementovaná v jazyku Python a je veľmi jednoducho použiteľná. Keďže OVS nie je implementovaný v jazyku Python, je nutné používať tieto generovacie skripty externe z OVS smerovača.

Samotná implementácia obsahuje viacero modulov, ktoré budú bežať ako samostatné procesy. Detailný opis modulov a ďalších skriptov:

- **assocProcess.py** - proces asociácie, a teda odpovedanie na asociačné požiadavky.
- **authProcess.py** - proces autentifikácie, a teda odpovedanie na autentifikačné požiadavky.
- **constants.py** - konštanty použité v projekte (SSID, adresy MAC, IP, ...)
- **createVethPair.sh** - bash skript, ktorý vytvorí dvojicu virtuálnych rozhraní potrebných pre komunikáciu s OVS.
- **manuf.txt** - súbor s výrobcami a ich MAC adresami.
- **monitorMode.sh** - bash skript, ktorý zmení mód bezdrôtového sieťového rozhrania na monitorovací.
- **probeProcess.py** - proces, ktorý odpovedá na *Probe* požiadavky.
- **receiveProcess.py** - proces, ktorý prijíma rámce z bezdrôtovej siete, zaobalí ich do Ethernet hlavičky a odošle OVS.
- **sendBeacons.py** - proces, ktorý odosiela *Beacon* rámce do bezdrôtovej siete.
- **sendProbeRequest.py** - skript, pre vytváranie a posielanie *Probe* požiadaviek do bezdrôtovej siete.
- **sendProcess.py** - proces, ktorý prijíma rámce od OVS, odstráni Ethernet hlavičku a pošle ich do bezdrôtovej siete.

# OpenFlow

## Úvod

OpenFlow je otvorený štandard, ktorý umožňuje testovanie experimentálnych protokolov pre výskum v počítačových sieťach, ktoré každý deň používame. OpenFlow je implementovaný ako súčasť komerčných Ethernet prepínačov, smerovačov a bezdrôtových prístupových bodov. OpenFlow poskytuje štandardizáciu umožňujúcu výskumným pracovníkom robiť experimenty bez toho, aby potrebovali od výrobcov interné informácie o svojich sieťových zariadeniach.

## Fungovanie

V klasickom smerovači alebo prepínači, dátové a kontrolné rámce (BGP, OSPF, ...) prechádzajú cez rovnaké sieťové zariadenie. OpenFlow prepínač tieto dve funkcionality rozdeľuje. Dátová časť stále prechádza cez prepínač, ale kontrolné rámce sú presunuté na oddelený kontroler, ktorý je typicky štandardný server. OpenFlow prepínač a kontroler komunikujú pomocou OpenFlow protokolu, ktorý definuje správy o tom aký paket bol prijatý, aký sa má odoslať, modifikáciách smerovacej tabuľky a štatistikách.

Dátová časť OpenFlow prepínača predstavuje tabuľkovú abstrakciu toku dát. Každý záznam tabuľky toku obsahuje súbor paketových polí, ktoré na základe zhodnosti s rámcom evokujú akciu rámca (odošli, modifikuj pole alebo zahod'). Keď OpenFlow prepínač prijme paket, ktorý nepozná a teda nemá žiadny záznam v tabuľke toku, pošle paket kontroleru. Kontroler potom rozhodne čo sa s paketom stane. Môže paket zahodiť alebo pridať záznam do tabuľky toku, ktorý povie prepínaču ako má takéto pakety smerovať.

## Možnosti

OpenFlow umožňuje jednoduché implementovanie inovatívnych smerovacích a prepínacích protokolov v počítačových sieťach. Je používaný napr.: vo virtuálnych riešeniach, vo vysoko-zabezpečených sieťach a v sieťach novej generácie založených na ip mobilných sieťach.

V našom prípade potrebuje rozšíriť o správy ktoré potrebujeme pre komunikáciu medzi wifi rozhraním a OVS. Na toto nám OpenFlow dovoľuje definovať vlastné OpenFlow správy.

Ak chceme definovania OpenFlow správ do prostredia OVS postupujeme nasledovným spôsobom. Do súboru v lib/ofp-msgs.h v štruktúre ofpraw pridáme komentár v definovanom štýle ako vidíme na nasledujúcom Obr. 2.:

```
/* ONFT 1.3 (2300): struct ofpl4_bundle_ctrl_msg, uint8_t[8][]. */  
OFPRAW_ONFT13_BUNDLE_CONTROL,
```

## Obr. 2. Definovanie OpenFlow správy v OVS

Tento komentár určuje že pridávame správu typu ONFT (Open Networking Foundation Extension message), platí že protokol Openflow je verzie 1.3, ID tejto správy je 2300. Za celou hlavičkou bude nasledovať štruktúra ofp12\_bundle\_ctrl\_message a 8 bitvy uint.

To ako sa má správa spracovať (parsovať) je určené v súbore lib/ofp-parse.c a v súbore lib/ofp-print.c. Príklad výpisu správy vidíme na nasledujúcom obrázku Obr. 3.:

```
static void
ofp_print_table_desc_reply(struct ds *s, const struct ofp_header *oh)
{
    struct ofpbuf b = ofpbuf_const_initializer(oh, ntohs(oh->length));
    for (;;) {
        struct ofputil_table_desc td;
        int retval;

        retval = ofputil_decode_table_desc(&b, &td, oh->version);
        if (retval) {
            if (retval != EOF) {
                ofp_print_error(s, retval);
            }
            return;
        }
        ofp_print_table_desc(s, &td);
    }
}
```

Obr. 3. Ukážka práce so správou v lib/ofp-print.c

# Floodlight

## Úvod

Floodlight je otvorený OpenFlow kontroler, ktorý je založený na Jave s Apache licenciou. Je podporovaný komunitou developerov a inžinierov z Big Switch Networks

Floodlight je navrhnutý, aby mohol pracovať s veľkým množstvom prepínačov, smerovačov a prístupových bodov, ktoré podporujú štandard OpenFlow protokolu. V našom projekte používame verziu 1.2.

## Možnosti

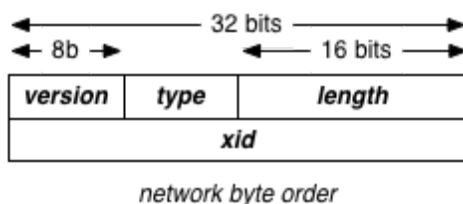
Tento kontroler dokáže spolupracovať s veľkým rozpätím virtuálnych aj fyzických OpenFlow prepínačov. Takisto je možné ho použiť v heterogénnej počítačovej sieti, a teda v sieti, kde sú použité aj OpenFlow aj non-OpenFlow prepínače. Je navrhnutý, aby pracoval s veľkým množstvom prepínačov, a teda je to multithreadový kontroler.

## Rozšírenia

Do SDN kontrolera sme pridali metódy na handlovanie našich experimenter custom správ. Po odchytení ich vieme následne spracovať a vykonať potrebné úkony. Takisto pribudli aj metódy na jednoduché vytváranie a odosielanie custom správ smerom z SDN kontrolera na OVS a taktiež na LVAP server. Odosielanie potrebných informácií na LVAP prebieha pomocou jednoduchšej HTTP komunikácie SERVER - CLIENT.

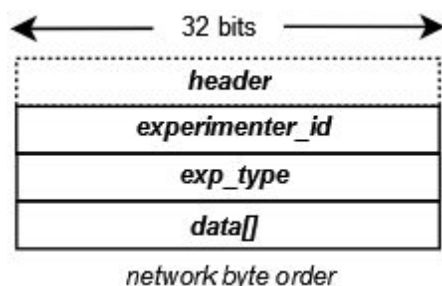
## Vlastné OpenFlow správy

Na komunikáciu medzi floodlight kontrolérom a Open VSwitchom sa používa Open Flow protokol. Floodlight kontroler má zadefinované viacero typov správ. Ide napríklad o Hello alebo Echo Request/Replay. OpenFlow obsahuje aj typ správy Experimenter. Tento typ správy slúži na vlastné definovanie správ. Dáva priestor vývojarom pridávať nové funkcionality pod takýmto typom správ.



Obr. 4. - OpenFlow hlavička

OpenFlow protocol verzia 1.4 má zadefinovaný tvar hlavičky. Prvé pole obsahuje verziu openflow. Typ id typu správy. Ak chceme nastaviť typ správy na Experimenter, tak typ sa rovná 4. Length obsahuje dĺžku správy a xid je id komunikačného kanála.



Obr. 5. - Experimenter hlavička

Experimenter správa má okrem bežnej OpenFlow hlavičky definovanú aj vlastnú. Osahuje *experimenter\_id* - číselnu hodnotu označujúcu id správy. Každý vlastný typ experimenter správy má svoj zadaný *experimenter\_id*. Dáta sú už konkrétne údaje, ktoré správa obsahuje.

Obrázky nachádzajúce sa nižšie zobrazujú nami definované správy. Tie sa však časom o trochu zmenili. Zmena nastala v tom, že všetky nami definované správy majú rovnaké experimenter ID. Ide o konvenciu pri implementovaní novej funkcionality. Rozdiel medzi jednotlivými správami z hľadiska označenia rieši premenná subtype, ktorá nadobúda pre každý typ našej správy unikátne ID.

Message name	Experimenter type - ID msg type	Payload parameters	The direction of sending the report	Description
--------------	---------------------------------	--------------------	-------------------------------------	-------------

Configuration message	0x00000000	Channel = 1B Length_SSID = 1B SSID = ??B	SDN controller -> WTP	Configure settings to WTP
-----------------------	------------	------------------------------------------------	-----------------------	---------------------------

Probe information message	0x00000001	MAC_STA = 6B	WTP -> SDN controller	Receipt Information Receive requests with the same SSID
---------------------------	------------	--------------	-----------------------	------------------------------------------------------------

Association information message	0x00000002	MAC_STA = 6B STATUS_CODE = 2B	WTP -> SDN controller	Associated station information
---------------------------------	------------	----------------------------------	-----------------------	--------------------------------

Add VAP context	0x00000004	BSSID = 6B MAC_STA = 6B IP_STA = 4B SSID_length= 1B SSID = Variable Length	SDN controller -> WTP	Adding a VAP context to WTP. The IP address may not be known depending on whether it is primarily to add a VAP context or to move a station to another WTP. When first added, the IP address has a zero value.
-----------------	------------	----------------------------------------------------------------------------------------	-----------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Remove VAP context	0x00000005	BSSID = 6B MAC_STA = 6B	SDN controller -> WTP	Removing the VAP context from WTP
--------------------	------------	----------------------------	-----------------------	-----------------------------------

Na definovanie nových typov openflow správ floodlight využíva Loxigen. Loxigen je projekt, ktorý slúži na generovanie knižníc s openflow metódami a definovanými správami. Najskôr je potrebné zadať, aké jednotlivé polia má správa obsahovať. Následne potrebné spustiť v príkazovom riadku v loxigen súbore *make package-java*, čo vygeneruje v loxigene v priečinku target balíky knižníc, ktoré je potrebné importovať do floodlight projektu. Na obrázku nižšie je uvedená implementácia z loxigénu.

## Add Vap Context msg

```
#version 5

struct of_add_vap_context_msg : of_mwga_header {
    uint8_t version;
    uint8_t type == 4;
    uint16_t length;
    uint32_t xid;
    uint32_t experimenter == 0x0005;
    uint32_t subtype == 4;
    uint64_t BSSID;
    uint64_t MAC_STA;
    uint32_t IP_STA;
    uint8_t Length_SSID;
    uint128_t SSID;
};
```

## Association Information msg

```
#version 5

struct of_association_information_msg : of_mwga_header {
    uint8_t version;
    uint8_t type == 4;
    uint16_t length;
    uint32_t xid;
    uint32_t experimenter == 0x0005;
    uint32_t subtype == 2;
    uint64_t MAC_STA;
    uint16_t STATUS_CODE;
};
```

## Configuration msg

```
#version 5

struct of_configuration_msg : of_mwga_header {
    uint8_t version;
    uint8_t type == 4;
    uint16_t length;
    uint32_t xid;
    uint32_t experimenter == 0x0005;
    uint32_t subtype == 0;
    uint8_t Channel;
    uint8_t Length_SSID;
    uint64_t SSID;
};
```



## Probe Information msg

```
#version 5
```

```
struct of_probe_information_msg : of_mwga_header {  
    uint8_t version;  
    uint8_t type == 4;  
    uint16_t length;  
    uint32_t xid;  
    uint32_t experimenter == 0x0005;  
    uint32_t subtype == 1;  
    uint64_t MAC_STA;  
};
```

```
struct of_mwga_header : of_experimenter {  
    uint8_t version;  
    uint8_t type == 4;  
    uint16_t length;  
    uint32_t xid;  
    uint32_t experimenter == 0x0005;  
    uint32_t subtype == ?;  
};
```

```
struct of_action_mwga : of_action_experimenter {  
    uint16_t type == 65535;  
    uint16_t len;  
    uint32_t experimenter == 0x0005;  
    uint16_t subtype == ?;  
    pad(20);  
};
```

## Remove Vap Context msg

```
#version 5

struct of_remove_vap_context_msg : of_mwga_header {
    uint8_t version;
    uint8_t type == 4;
    uint16_t length;
    uint32_t xid;
    uint32_t experimenter == 0x0005;
    uint32_t subtype == 5;
    uint64_t BSSID ;
    uint64_t MAC_STA;
};
```

## MWGA header and action struct

```
#version 5

struct of_mwga_header : of_experimenter {
    uint8_t version;
    uint8_t type == 4;
    uint16_t length;
    uint32_t xid;
    uint32_t experimenter == 0x0005;
    uint32_t subtype == ?;
};

struct of_action_mwga : of_action_experimenter {
    uint16_t type == 65535;
    uint16_t len;
    uint32_t experimenter == 0x0005;
    uint16_t subtype == ?;
    pad(20);
};
```

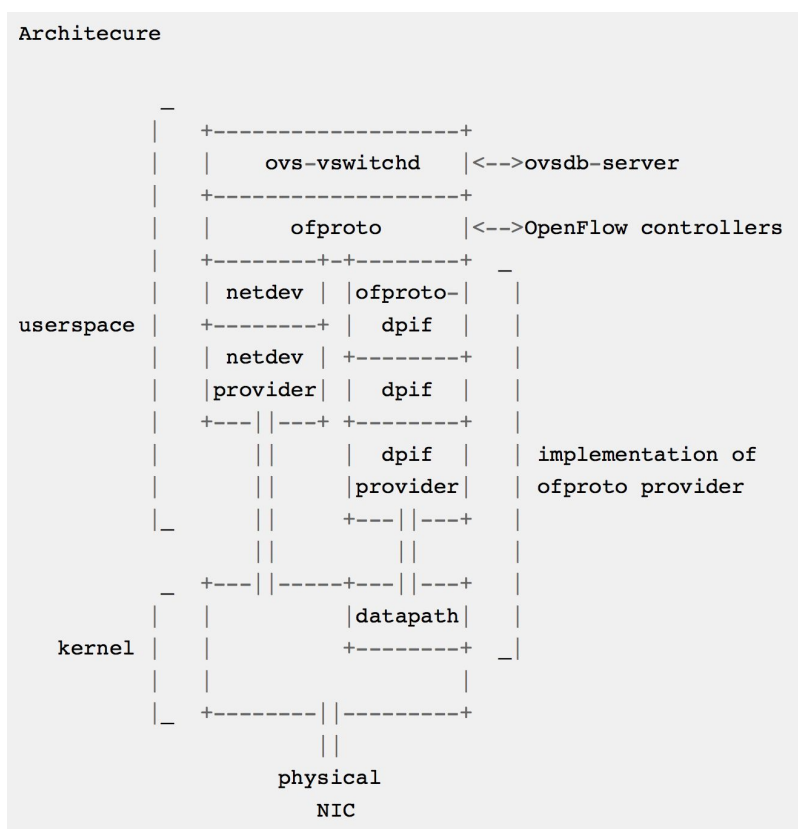
# Open vSwitch

## Úvod

Open vSwitch alebo aj OVS je otvorená implementácia distribuovaného multivrstvového prepínača, ktorý má licenciu Apache 2.0. Je dizajnovaný pre automatizáciu veľkých počítačových sietí pomocou softvérových rozšírení. OVS podporuje, okrem iných, aj protokol OpenFlow, ktorý používame v našom projekte. Je implementovaný v jazyku C, ktorý pomáha lepšej prenosnosti medzi systémami a obsahuje implementáciu do linuxového jadra. OVS bol implementovaný do rôznych riešení, napr.: OpenStack, openQRM, OpenNebula atď.

## Možnosti

OVS obsahuje podporu pre VLAN siete, IGMP, LLDP, SPBM, BFD, RSTP, GRE, IPv6 a veľa ďalších protokolov. Obsahuje takisto podporu QoS, OpenFlow protokolu a implementácie pre jazyky C aj Python, a “pipelining” engine pre cache-ovanie viacero smerovacích tabuliek



Obr. 4. Architektúra OVS

## Pridanie custom OF správ do OVS

Pre pridanie nových správ je potrebné ich zdefinovať v súbore

**include/openvswitch/ofp-msgs.h** v preddefinovanom formáte a následne spúšťať build a odstraňovať všetky upozornenia a chyby. Správa sa definuje v enum ofpraw v nasledujúcom formáte:

```
/*  
 * MWGA  
 */  
/* MWGA 1.4 (4): struct of_add_vap_context_msg. */  
OFPRAW_MWGA_ADD_VAP_CONTEXT,  
  
/* MWGA 1.4 (5): struct of_remove_vap_context_msg. */  
OFPRAW_MWGA_REMOVE_VAP_CONTEXT,  
  
/* MWGA 1.4 (2): struct of_association_information_msg. */  
OFPRAW_MWGA_ASSOCIATION_INFORMATION,  
  
/* MWGA 1.4 (0): struct of_configuration_msg. */  
OFPRAW_MWGA_CONFIGURATION,  
  
/* MWGA 1.4 (1): struct of_probe_information_msg. */  
OFPRAW_MWGA_PROBE_INFO,
```

Obr. 5. Definícia OF správ v OVS

Pre príklad na vysvetlenie prvej správy:

**MWGA** - custom experimenter OF správa

**1.4** - podporovaná verzia OF

**(0)** - subtype

**Struct of\_configuration\_msg** - štruktúra správy

Následne je potrebné pridať túto správu do enum ofptype (pre prípad ak má správa viac konkrétnych formátov) :

```
enum ofptype {
    /* MWGA */
    OFPTYPE_ADD_VAP_CONTEXT, /* OFPRAW MWGA ADD VAP CONTEXT. */
    OFPTYPE_REMOVE_VAP_CONTEXT, /* OFPRAW MWGA REMOVE VAP CONTEXT */
    OFPTYPE_ASSOCIATION_INFORMATION, /* OFPRAW MWGA ASSOCIATION_INFORMATION */
    OFPTYPE_CONFIGURATION, /* OFPRAW MWGA CONFIGURATION */
    OFPTYPE_PROBE_INFO, /* OFPRAW MWGA PROBE_INFO */
}
```

Obr. 6. Pridanie správy do enumofptype

Aby bola takto zadefinovaná správa je potrebe pridať do súboru **build-aux/extract-ofp-msgs** do funkcie `extract_ofp_msgs` vetvu pre vyriešenie definovaných správ MWGA:

```
elif type_ == 'MWGA':
    hdrs = (version, OFPT_VENDOR, 0, MWGA_VENDOR_ID, number)
```

Obr. 7. Rozoznávanie zadefinovaných MWGA OF správ

Aby bol OVS kompilovateľný je potrebné zadefinovať štruktúry, ktoré tieto definície správ potrebujú. Definície štruktúr treba pridať do súboru: **include/openflow/openflow-1.4.h** keďže pracujeme s verziou openflow 1.4. Štruktúry sú zadefinované nasledovne:

```
/*
 * MWGA
 */

struct of_add_vap_context_msg{
    uint8_t version;
    uint8_t type;
    uint8_t length[2];
    uint8_t xid[4];
    uint8_t experimenter[4];
    uint8_t subtype[4];
    uint8_t BSSID[8];
    uint8_t MAC_STA[8];
    uint8_t IP_STA[4];
    uint8_t Length_SSID;
    uint8_t SSID[16];
    uint8_t padding[3];
};
```

```

struct of_configuration_msg {
    uint8_t version;
    uint8_t type;
    uint8_t length[2];
    uint8_t xid[4];
    uint8_t experimenter[4];
    uint8_t subtype[4];
    uint8_t Channel;
    uint8_t Length_SSID;
    uint8_t SSID[8];
    uint8_t padding[2];
};

struct of_probe_information_msg {
    uint8_t version;
    uint8_t type;
    uint8_t length[2];
    uint8_t xid[4];
    uint8_t experimenter[4];
    uint8_t subtype[4];
    uint8_t MAC_STA[8];
};

struct of_remove_vap_context_msg {
    uint8_t version;
    uint8_t type;
    uint8_t length[2];
    uint8_t xid[4];
    uint8_t experimenter[4];
    uint8_t subtype[4];
    uint8_t BSSID[8];
    uint8_t MAC_STA[8];
};

struct of_association_information_msg {
    uint8_t version;
    uint8_t type;
    uint8_t length[2];
    uint8_t xid[4];
    uint8_t experimenter[4];
    uint8_t subtype[4];
    uint8_t MAC_STA[8];
    uint8_t STATUS_CODE[2];
    uint8_t padding[2];
};

```

Obr. 8. Struktúry OF správ



Keď je všetko skompilované bez upozornení a chýb, uvidíme tieto raw správy v súbore `/lib/ofp-msgs.inc` ako na nasledujúcom obrázku:

```
static struct raw_instance ofpraw_mwga add_vap_context_instances[] = {
    { {0, NULL}, {5, 4, 0, 0x6, 4}, OFPRAW_MWGA_ADD_VAP_CONTEXT, 0 },
};
static struct raw_instance ofpraw_mwga remove_vap_context_instances[] = {
    { {0, NULL}, {5, 4, 0, 0x6, 5}, OFPRAW_MWGA_REMOVE_VAP_CONTEXT, 0 },
};
static struct raw_instance ofpraw_mwga association_information_instances[] = {
    { {0, NULL}, {5, 4, 0, 0x6, 2}, OFPRAW_MWGA_ASSOCIATION_INFORMATION, 0 },
};
static struct raw_instance ofpraw_mwga configuration_instances[] = {
    { {0, NULL}, {5, 4, 0, 0x6, 0}, OFPRAW_MWGA_CONFIGURATION, 0 },
};
static struct raw_instance ofpraw_mwga probe_info_instances[] = {
    { {0, NULL}, {5, 4, 0, 0x6, 1}, OFPRAW_MWGA_PROBE_INFO, 0 },
};
```

Obr. 8. Raw OF správy

# Modul pre manažment LVAP

## LVAP manažment server

Modul LVAP server zabezpečuje manažment LVAP štruktúr. LVAP štruktúry špecifikujú jednotlivé stanice pripojené k AP a ich parametre. LVAP štruktúra pre individuálne stanice pozostáva z nasledovných atribútov :

- IP adresa stanice
- MAC adresa stanice
- BSSID stanice
- SSID bezdrôtovej siete
- WTP identifikátor

LVAP štruktúra je implementovaná ako samostatná vnhiezdená statická trieda v jazyku Java. Každý objekt triedy pre LVAP štruktúry obsahuje príslušné spomenuté atribúty a je vytváraný konštruktorom.

Jednotlivé LVAP štruktúry sú uložené v databáze spravujúcej VAP kontext. LVAP server disponuje časťou, ktorá poskytuje základnú CRUD funkcionality pre komunikáciu s touto databázou.

Okrem časti pre komunikáciu s databázou obsahuje LVAP server časť pre komunikáciu s klientom a modul pre generovanie unikátnych BSSID jednotlivých staníc. Každá z týchto častí je podrobnejšie opísaná v nasledovných podkapitolách.

## Komunikácia s databázou

Pre fungujúci manažment VAP kontextu, presnejšie LVAP štruktúr je potrebné, aby tieto štruktúry boli centrálné uložené v databáze. LVAP server komunikuje s Java Derby databázou `lvap_struct` a pracuje s jej tabuľkou `STRUCTURES`, v ktorej sú spomenuté štruktúry uložené.

Tabuľka `STRUCTURES` disponuje nasledovnými stĺpcami :

- IP - typu `VARCHAR` - obsahuje IP adresu stanice
- MAC - typu `BLOB` - obsahuje MAC adresu stanice
- BSSID - typu `BLOB` - obsahuje vygenerované BSSID stanice
- SSID - typu `VARCHAR` - obsahuje SSID bezdrôtovej siete
- WTP - typu `INTEGER` - obsahuje identifikátor WTP

LVAP server obsahuje metódy pre vytvorenie pripojenia k databáze, pre ukončenie pripojenia k databáze, pre vyhľadanie záznamu v databáze. Veľmi dôležitou metódou je metóda `checkMAC()`, ktorá ako parameter prijíma reťazec MAC adresy stanice a kontroluje, či už sa stanica s danou MAC adresou v databáze nachádza. Všetky MAC adresy v databáze musia byť unikátne. Na základe výsledku, ktorý vráti metóda `checkMAC()` - ide o



boolean odpoveď TRUE alebo FALSE - sa modifikuje odpoveď, ktorú LVAP server pošle naspäť klientovi. Taktiež závisí, či LVAP server pridá ďalší záznam do tabuľky databázy. Nový záznam sa pridáva iba v prípade, že tabuľka ešte neobsahuje záznam s MAC adresou, ktorú poslal klient v požiadavke LVAP serveru.

## Komunikácia s klientom

LVAP server komunikuje s klientom na porte 8080. Na tomto porte prijíma LVAP server požiadavky klienta. Požiadavka klienta prichádza na server vo formáte reťazca, ktorý definuje MAC adresu stanice.

Po prijatí požiadavky klienta LVAP server túto požiadavku nasledovne spracuje. Server vykoná pripojenie na databázu a skontroluje záznamy tabuľky, konkrétne či už sa klientom špecifikovaná MAC adresa v tabuľke nachádza. V prípade, že hľadanie záznamu v tabuľke vrátilo kladnú odpoveď, LVAP server klientovi pošle naspäť odpoveď vo forme reťazca "Y" a ďalšie operácie nevykonáva, pokiaľ nedostane ďalšiu požiadavku.

V prípade, že MAC adresa špecifikovaná klientom sa v tabuľke ešte nenachádza, úlohou LVAP servera je záznam s touto MAC adresou prijať. LVAP server vytvorí kompletnú LVAP štruktúru zodpovedajúcu danej MAC adrese a túto štruktúru pridá do tabuľky záznamov. Klientovi následne pošle odpoveď vo forme reťazca "Y".

## Generovanie BSSID

Jednou zo základných funkcionalít nami pridávaného rozšírenia do Floodlight SDN Controllera bolo pridanie komponentu, ktorý bude zabezpečovať generovanie unikátnych BSSID pre jednotlivé stanice v sieti. V kontexte IEEE štandardu bezdrôtových sietí predstavuje BSSID Basic Service Set Identification, z praktického hľadiska BSSID predstavuje MAC adresu bezdrôtového access pointu (WAP). Úlohou Access Pointu (ďalej len AP), je vytvoriť ilúziu existencie samostatnej inštancie AP pre každú pripojenú stanicu.

Komponent zabezpečujúci generovanie unikátnych BSSID pre pripojené stanice bol implementovaný ako samostatná Java trieda do existujúceho Floodlight projektu. Pre zabezpečenie unikátnosti generovaných BSSID bol na generovanie použitý hashovací algoritmus SHA-1, mierne upravený pre použitie na naše účely.

Samotná funkcionalita generovania v komponente, o ktorý sme Floodlight rozšírili, je obsiahnutá vo funkcii `getUniqueBSSID()`, ktorá ako parameter prijíma MAC adresu stanice, ktorá sa chce pripojiť na AP. Následne je MAC adresa konvertovaná na pole bajtov. Získané pole bajtov je hashované algoritmom SHA-1, ktorého implementáciu sme získali z Java knižnice `MessageDigest`.

Výstupom SHA-1 algoritmu je 20 bajtový fingerprint. Keďže BSSID je ďalej používané na adresovanie v sieti, nie je možné použiť 20 bajtový fingerprint, ale zo získaného fingerprintu použiť prvých 6 bajtov. Okrem tohoto zásahu do fingerprintu sa taktiež vykonáva

modifikácia prvých dvoch bajtov BSSID, ktoré sú pevne nastavené na hodnotu “ee:ee” a to z dôvodu zvýšenia viditeľnosti paketov pri odchyťovaní vo Wiresharku. Faktom je, že zásahom do SHA-1 fingerprintu sa síce na jednej strane zvyšuje riziko kolízií hashov a tým pádom teoreticky aj vygenerovaných BSSID, avšak po dôkladnej diskusii v tíme sme toto riziko stanovili na minimálne.

```
//create SHA1 fingerprint & convert to byte array & back to MacAddress
try {
    MessageDigest md = MessageDigest.getInstance("SHA-1");
    md.update(buffer);
    byte[] digest = Arrays.copyOfRange(md.digest(),0,6); //take the first 6 bytes from the fingerprint

    //overwrite specific bytes in byte array with byte value of 'e' ...hork's wish
    digest[0] = tmp_byte[0];
    digest[1] = tmp_byte[1];

    BSSID = MacAddress.of(digest);
} catch (NoSuchAlgorithmException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Obr. 3. - ukážka kódu BSSID generátora

## Prílohy

### A Metodiky

# Metodika dokumentácie

Bc. Katarína Bedejová

Názov tímu : MWGA - Make Wifi Great Again

Vedúci tímu: Ing. Rastislav Bencel

Členovia tímu: Bc.Matej Belluš

Bc.Marcel Sabol,

Bc.Michal Goffstein

Bc. Michal Hampel

Bc. Marián Heseck,

Pracovisko: FIIT STU - 5.45

Akademický rok: 2017/2018

Dátum odovzdania: 11.5.2018

# Obsah

<b>Úvod</b>	<b>3</b>
<b>Metodika generovania dokumentácie</b>	<b>4</b>
Floodlight	4
Základné dokumentačné tagy	4
Opis dokumentačných tagov s príkladmi ich použitia	4
OpenvSwitch	7
<b>Metodika tvorby dokumentácie</b>	<b>9</b>
Pravidlá formátu tvorenej dokumentácie	9

# 1.Úvod

Dokumentácia ku kódu je dôležitá predovšetkým z dôvodu udržiavateľnosti - napríklad aby bolo umožnené naštudovať si, aká časť kódu poskytuje akú funkcionálnu hodnotu ľuďom, ktorí na projekte budú v budúcnosti pokračovať. Dokumentácia ku kódu pokrýva názvy funkcií, ich význam/účel, a taktiež požadované vstupy a nimi poskytované výstupy. Okrem toho metodika dokumentácie definuje akým spôsobom sa budú vytvárať iné druhy dokumentácií, a to konkrétne tie, ktoré nebudú generované z kódu, ale tie, čo budú vytvárané samostatne. Pravidlá formy dokumentácie sú určené nato, aby nami vytváraná dokumentácia bola vždy jednotná a tým pádom aj prehľadná a ľahko čitateľná.

## 2. Metodika generovania dokumentácie

V tejto kapitole je podrobne opísaná metodika generovania dokumentácie.

### Floodlight

Čo sa týka zdrojového kódu projektu Floodlight, napísaný je v jazyku Java a pre generovanie dokumentácie zdrojového kódu sa používa knižnica javadoc.

#### Základné dokumentačné tagy

- `@author`
- `@code`
- `@docRoot`
- `@deprecated`
- `@exception`
- `@inheritDoc`
- `@link`
- `@linkplain`
- `@param`
- `@return`
- `@see`
- `@serial`
- `@serialData`
- `@serialField`
- `@since`
- `@throws`
- `@value`
- `@version`

#### Opis dokumentačných tagov s príkladmi ich použitia

`@author`

Pridá autora triedy

Syntax : `@author` "meno-autora"

`@code`

Zobrazuje text tela kódu v kódovom fonte.

Syntax: `{@code text}`

#### @docRoot

Reprezentuje relatívnu cestu do root adresára, kde sa vygeneruje dokumentácia.

Syntax: {@docRoot}

#### @deprecated

Označuje deprecated API (Api, ktoré by sme už nemali používať)

Syntax: @deprecated "text"

#### @exception

Do generovanej dokumentácie pridá Throws podnadpis. Synonymum k tagu @throws.

Syntax: @exception "class-name popis"

#### @link

Do dokumentácie pridáva link, ktorý referuje na určené miesto v dokumentácii - môže to byť popis balíka, triedy, metódy, atď. Link zobrazí v kódovom fonte.

Syntax: {@link package.class#member label}

#### @linkplain

Do dokumentácie pridáva link, ktorý referuje na určené miesto v dokumentácii - môže to byť popis balíka, triedy, metódy, atď. Link zobrazí v plain fonte.

Syntax: {@linkplain package.class#member label}

#### @param

Do dokumentácie pridáva sekciu parameter.

Syntax: @param parameter-name description

#### @return

Do dokumentácie pridáva sekciu return.

Syntax: @return description

#### @see

Pridáva See Also nadpis, ktorý referencuje na určenú sekciu v dokumentácii.

Syntax: @see reference

#### @serial

Komentár v dokumentácii pre serializovateľné pole.

Syntax: @serial field-description | include | exclude

#### @serialData

Popisuje dáta písané funkciami writeObject( ) or writeExternal( ).

Syntax: @serialData data-description

#### @serialField

Popisuje komponent typu ObjectOutputStreamField.

Syntax: @serialField field-name field-type field-description

### @since

Pridáva podnadpis Since, ktorý referencuje na určenú sekciu v generovanej dokumentácii, ktorá obsahuje text s informáciami k releasu.

Syntax: @since release

### @throws

Do generovanej dokumentácie pridá Throws podnadpis. Synonymum k tagu @exception.

Syntax: @throws class-name description

### @value

Špecifikuje hodnotu určenej konštanty.

Syntax: {@value package.class#field}

### @version

Pridáva podnadpisVersion, ktorý referencuje na určenú sekciu v generovanej dokumentácii, ktorá obsahuje text s informáciami k verzii.

Syntax: @version version-text

Generovanie javadoc dokumentácie je definované v súbore build.xml v projekte Floodlight :

```
321
322     <target name="javadoc">
323         <javadoc access="protected"
324             author="true"
325             classpathref="classpath"
326             destdir="${docs}"
327             doctitle="Floodlight"
328             nodeprecated="false"
329             nodeprecatedlist="false"
330             noindex="false"
331             nonavbar="false"
332             notree="false"
333             source="1.7"
334             sourcepath="${source}"
335             splitindex="true"
336             use="true"
337             version="true"/>
338     </target>
339
```



# OpenvSwitch

Generovanie dokumentácie pre OpenvSwitch je definované v súbore Automake.mk a zahŕňa nasledovné .rst súbory. Generovanie dokumentácie priamo zo zdrojového kódu ako pri Floodlighte je neexistujúce.

- Documentation/group-selection-method-property.txt \
- Documentation/\_static/logo.png \
- Documentation/\_static/overview.png \
- Documentation/conf.py \
- Documentation/index.rst \
- Documentation/contents.rst \
- Documentation/intro/index.rst \
- Documentation/intro/what-is-ovs.rst \
- Documentation/intro/why-ovs.rst \
- Documentation/intro/install/index.rst \
- Documentation/intro/install/bash-completion.rst \
- Documentation/intro/install/debian.rst \
- Documentation/intro/install/documentation.rst \
- Documentation/intro/install/distributions.rst \
- Documentation/intro/install/dpdk.rst \
- Documentation/intro/install/fedora.rst \
- Documentation/intro/install/general.rst \
- Documentation/intro/install/netbsd.rst \
- Documentation/intro/install/ovn-upgrades.rst \
- Documentation/intro/install/rhel.rst \
- Documentation/intro/install/userspace.rst \
- Documentation/intro/install/windows.rst \
- Documentation/intro/install/xenserver.rst \
- Documentation/tutorials/index.rst \
- Documentation/tutorials/ovs-advanced.rst \
- Documentation/tutorials/ovn-openstack.rst \
- Documentation/tutorials/ovn-sandbox.rst \
- Documentation/topics/index.rst \
- Documentation/topics/bonding.rst \
- Documentation/topics/idl-compound-indexes.rst \
- Documentation/topics/datapath.rst \
- Documentation/topics/design.rst \
- Documentation/topics/dpdk/index.rst \
- Documentation/topics/dpdk/ring.rst \
- Documentation/topics/dpdk/vhost-user.rst \
- Documentation/topics/testing.rst \
- Documentation/topics/high-availability.rst \
- Documentation/topics/integration.rst \
- Documentation/topics/language-bindings.rst \
- Documentation/topics/openflow.rst \
- Documentation/topics/ovsdb-replication.rst \
- Documentation/topics/porting.rst \
- Documentation/topics/tracing.rst \
- Documentation/topics/windows.rst \
- Documentation/howto/index.rst \

Documentation/howto/docker.rst \  
Documentation/howto/dpdk.rst \  
Documentation/howto/kvm.rst \  
Documentation/howto/libvirt.rst \  
Documentation/howto/selinux.rst \  
Documentation/howto/ssl.rst \  
Documentation/howto/lisp.rst \  
Documentation/howto/openstack-containers.rst \  
Documentation/howto/qos.png \  
Documentation/howto/qos.rst \  
Documentation/howto/sflow.png \  
Documentation/howto/sflow.rst \  
Documentation/howto/tunneling.png \  
Documentation/howto/tunneling.rst \  
Documentation/howto/userspace-tunneling.rst \  
Documentation/howto/vlan.png \  
Documentation/howto/vlan.rst \  
Documentation/howto/vtep.rst \  
Documentation/ref/index.rst \  
Documentation/faq/index.rst \  
Documentation/faq/configuration.rst \  
Documentation/faq/contributing.rst \  
Documentation/faq/design.rst \  
Documentation/faq/general.rst \  
Documentation/faq/issues.rst \  
Documentation/faq/openflow.rst \  
Documentation/faq/qos.rst \  
Documentation/faq/releases.rst \  
Documentation/faq/terminology.rst \  
Documentation/faq/vlan.rst \  
Documentation/faq/vxlan.rst \  
Documentation/internals/index.rst \  
Documentation/internals/authors.rst \  
Documentation/internals/bugs.rst \  
Documentation/internals/committer-grant-revocation.rst \  
Documentation/internals/committer-responsibilities.rst \  
Documentation/internals/documentation.rst \  
Documentation/internals/mailling-lists.rst \  
Documentation/internals/maintainers.rst \  
Documentation/internals/patchwork.rst \  
Documentation/internals/release-process.rst \  
Documentation/internals/security.rst \  
Documentation/internals/contributing/index.rst \  
Documentation/internals/contributing/backporting-patches.rst \  
Documentation/internals/contributing/coding-style.rst \  
Documentation/internals/contributing/coding-style-windows.rst \  
Documentation/internals/contributing/documentation-style.rst \  
Documentation/internals/contributing/libopenvswitch-abi.rst \  
Documentation/internals/contributing/submitting-patches.rst \  
Documentation/requirements.txt \

## 3. Metodika tvorby dokumentácie

Táto kapitola sa venuje metodike tvorby dokumentácia. Do tejto dokumentácie je zahrnutá všetka dokumentácia, ktorá nie je generovaná zo zdrojového kódu projektu, ale je tým pádom písaná jednotlivými členmi nášho tímu.

### 3.1 Pravidlá formátu tvorenej dokumentácie

Pre text dokumentácie písanej jednotlivými členmi tímu sa používa písmo Arial veľkosti 11px. Hlavné nadpisy jednotlivých kapitolí dokumentácie sú typu 1 (Heading 1), podnadpisy sú typu 2 (Heading 2).

Číslovanie nadpisov sa používa v nasledovnom tvare :

1. Prvý nadpis (začiatok kapitoly)
  - 1.1 Prvý podnadpis (podkapitola)
  - 1.2 Druhý podnadpis (podkapitola)
2. Druhý nadpis (začiatok ďalšej kapitoly)
  - 2.1 Prvý podnadpis (podkapitola)
  - 2.2 Druhý podnadpis (podkapitola)

Každý samostatný appendix musí mať uvedený názov tímu, názov dokumentácie, meno autora dokumentácie, meno vedúceho projektu, mená členov tímu, správny akademický rok, miesto pracoviska a dátum odovzdania.

Každá dokumentácia (prípadne appendix) musí mať vygenerovaný obsah.

V prípade, že sa v projekte vyskytujú nejaké cudzie slová alebo technické výrazy, ktorých význam nemusí byť všeobecne známy, je potrebné, aby použité cudzie slovo bolo vysvetlené v slovníku výrazov. V prípade, že sa použijú skratky, je potrebné, aby význam skratky bol vysvetlený v Použitých skratkách.

Pre uvádzanie bibliografických odkazov sa používa forma ISO 690 a ISO 690-2.

Výsledná forma dokumentácie musí byť exportovaná do formátu pdf.

# Metodika písania zdrojového kódu

Bc. Michal Hampel

Vedúci projektu: Ing. Rastislav Bencel

Názov tímu: MGWA - Make Wifi Great Again

Členovia tímu: Bc. Katarina Bedejová

Bc. Matej Belluš

Bc. Marcel Sabol

Bc. Michal Gottstein

Bc. Marián Heseck

Bc. Michal Hampel

Bc. Miroslav Procházka

Pracovisko: FIIT STU - 5.45

Akademický rok: 2017/2018

Posledná zmena: 15.11.2017

# Obsah

<b>1. Úvod</b>	<b>3</b>
<b>2. Metodika pomenování (naming conventions)</b>	<b>4</b>
Java	4
C	5
<b>3. Metodika komentářů</b>	<b>6</b>
Java	6
C	6

# 1. Úvod

V tejto metodike je za cieľ zjednotiť menné konvencia a formát zdrojového kódu napísaného v programovacom jazyku C alebo Java na jednotný štandardizovaný štýl. Takto naformátovaný zdrojový kód je v konečnom dôsledku jednoduchšie čitateľný a ľahšie pochopiteľný pre vývojárov. Keďže sa všetci členovia tímu podieľame na vývoji, tak táto metodika je určená pre celý tím.

## 2. Metodika pomenovaní (naming conventions)

Vo všeobecnosti platí, že názvy sa píšú po anglicky. Názov by mal čo najlepšie vystihovať objekt, funkciu, metódu, rozhranie ..., ktoré reprezentujú. Nemali by sa používať názvy, ktoré nemajú plný význam. A to napríklad jedno písmenkové názvy premenných s výnimkou premenných, ktoré slúžia ako ukazovatele na poradia v cykloch.

Nasledujúce tabuľky ukazujú pravidlá názvoslovia, ktoré je nutné dodržiavať:

### Java

Typ	Pravidlá názvoslovia	Príklad
Balíčky	balíčky nazývame vždy s malými písmenami anglickej abecedy. Mali by obsahovať názov domény v obrátenom poradí. lowercase	sk.stuba.fiit.mwga
Triedy	Trieda by mala obsahovať podstatné meno, ak sa názov skladá z viacerých slov každé začiatkové písmeno slova by malo byť veľké. Názvy sa píšú bez medzier. UpperCamelCase	class SavingAccount;
Rozhrania	Rozhranie má také isté pravidlá ako trieda. UpperCamelCase	interface BankAccount;
Metódy	Názov metódy by mal obsahovať sloveso. Pri viacslovných názvoch by malo byť prvé písmeno s malým každé ďalšie slovo s počiatočným veľkým písmenom. lowerCamelCase	getMethod();
Premenné	Názov premenej sa píše ako názov metódy. lowerCamelCase	SavingAccount mainAccount;
Konštanty	Názvy konštánt sa píšú s veľkými písmenami. Ak názov konštanty obsahuje viac slov sú oddelené podtržníkom (_). UPPERCASE	static final int MAX_WIDTH=1;

## C

Typ	Pravidlá názvoslovia	Príklad
Funkcie	Názvy by mali čo najviac zodpovedať tomu čo daná funkcia robí. Používajú sa malé písmená. Pri viacslovných názvoch sú slová oddelené podtržníkom (_).	check_for_access()
Premenné	Podobne ako používame názvoslovie pre funkcie používame aj pre premenné	Processor access_processor;
Ukazovateľ	* píšeme k názvu premennej, nie k typu	char *name;
Globálne premenné	Pred globalne premenné dávame prefix _g	Logger g_log;
Makrá	Názovy makier píšeme s veľkými písmenami. Pri viacslovných názvoch oddeľujeme podtržítom (_).	#define MAX(a,b) blah
Konštanty	Názvy konštánt sa píšu s veľkými písmenami. Ak názov konštanty obsahuje viac slov sú odelené podtržníkom (_). UPPERCASE	const int MAX_WIDTH=1;



### 3. Metodika komentárov

Programovacie jazyky zväčša obsahujú viac typov komentárov. Jazyk Java ponúka 3 typy komentárov a to jednoriadkový, viac riadkový a javadoc komentár. Z javadoc komentárov sa dá pomocou nástroja vytvoriť javadoc dokumentácia. Jazyk C ponúka 2 typy komentárov a to jedno riadkový a viac riadkový avšak automatické generovanie dokumentácie priamo nepodporuje.

Nasledujúce tabuľky prehľadné ukazujú ako a kde sa daný komentár má používať:

#### Java

Typ	Použitie	Príklad
Jednoriadkový	Používa sa pri jednoduchých poznámkach v kóde	//format meno_priezvisko
Viacriadkový	Používa sa pri sprehladnení čítania kódu kedy opisujeme funkcionality na viac riadkov	/* potrebný viacriadkový komentár ktorý opisuje čo robí blok kódu */
Javadoc	Používa sa keď chceme z kódu vygenerovať dokumentáciu. Píšu sa nad definíciami metód, tried alebo rozhraní. Dajú sa použiť aj notácie napr.: @param -parametre metódy @return návratová hodnota @since kedy bola napísaná @author autor	/** javadoc komentár * používa aj notácie napr. * @param name nejaké meno * @return vráti pekne meno */

#### C

Typ	Použitie	Príklad
Jednoriadkový	Používa sa pri jednoduchých poznámkach v kóde	//format meno_priezvisko
Viacriadkový	Používa sa pri sprehladnení čítania kódu kedy opisujeme funkcionality na viac riadkov	/* potrebný viacriadkový komentár ktorý opisuje čo robí blok kódu */

# Metodika komunikácie

Bc. Miroslav Procházka

Vedúci projektu: Ing. Rastislav Bencel  
Názov tímu: MGWA - Make Wifi Great Again  
Členovia tímu: Bc. Katarina Bedejová  
                  Bc. Matej Belluš  
                  Bc. Marcel Sabol  
                  Bc. Michal Gottstein  
                  Bc. Marián Hesek  
                  Bc. Michal Hampel  
                  Bc. Miroslav Procházka  
Pracovisko: FIIT STU - 5.45  
Akademický rok: 2017/2018  
Posledná zmena: 11.5.2018

# Obsah

<b>1.Úvod</b>	<b>3</b>
<b>2.Osobné stretnutia</b>	<b>4</b>
<b>3.Elektronické komunikačné kanály</b>	<b>5</b>

# 1. Úvod

Metodika komunikácie sprostredkúva spôsoby a prostriedky komunikácie medzi členmi tímu.

## 2. Osobné stretnutia

Osobné stretnutia sú každý utorok o 8:00 v miestnosti 5.45 v budove FIIT STU. Je možné si dohodnúť aj stretnutie v inom čase pomocou nižšie špecifikovaných elektronických komunikačných kanálov.

### 3. Elektronické komunikačné kanály

Táto kapitola obsahuje informácie o používaných elektronických informačných kanáloch členmi tímu. Patria sem:

- **Discord** - aplikácia pre rôzne platformy (MS Windows, Linux, Android, iOS, browser), ktorá slúži ako primárny komunikačný kanál. Na tímovom serveri je zadaných viacero miestností:
  - general - hlavná miestnosť bez primárneho určenia.
  - pinned - odkazy a zdroje, pre rýchly prístup k potrebným informáciám, nástrojom atď.
  - server - informácie o serveri, na ktorom je stránka.
  - sprint - komunikácia rámci aktuálneho šprintu.
  - bugy - upozorňovanie na aktuálne chyby.
  - help - kanál pre pomoc pri riešení úloh.
- **ZenHub** - komunikačný kanál vo forme riadenia projektu, ktorý je pripojený k verziovaciemu nástroju GitHub. Komunikácia spočíva v zadaní úloh do jednotlivých fáz, ich komentovanie a presúvanie medzi fázami.
- **Facebook** - sekundárny, slúži ako sekundárny nástroj pre komunikáciu, či už medzi jednotlivými členmi tímu alebo rámci celého tímu, pre ktorý je vytvorený skupinový chat - "Tímový projekt 2 17/18". Tento kanál sa používa pri zlyhaní primárneho komunikačného kanála.
- **Telefón** - používa sa na ako terciárny komunikačný kanál pre naliehavšie záležitosti.
- **Email** - používa sa ako notifikačný kanál, a teda pre notifikácie o pridelených úlohách, vytvorených dokumentoch atď.

# Metodika verziovania a prehliadok kódu

Bc. Matej Belluš

Vedúci projektu: Ing. Rastislav Bencel

Názov tímu: MGWA - Make Wifi Great Again

Členovia tímu: Bc. Katarina Bedejová

Bc. Matej Belluš

Bc. Marcel Sabol

Bc. Michal Gottstein

Bc. Marián Hesek

Bc. Michal Hampel

Bc. Miroslav Procházka

Pracovisko: FIIT STU - 5.45

Akademický rok: 2017/2018

Posledná zmena: 11.5.2018

# Obsah

<b>Úvod</b>	<b>3</b>
<b>Slovník pojmov - Github</b>	<b>3</b>
<b>Základný workflow</b>	<b>3</b>
<b>Code review</b>	<b>3</b>
<b>Obsah Github Wiki</b>	<b>4</b>
Consult some git tutorials:	4
Create new branch:	4
Example commit:	4
Example Push:	4
rollback a github repository to a specific commit	4



# Úvod

Na verziovanie kódu sa používa nástroj Github spolu s agilnou nadstavbou ZenHub. Cieľom tejto metodiky nie je naučiť niekoho prácu s gitom, ale ako postupovať.

## Slovník pojmov - Github

- **Commit** - jasne označuje stav kódu v čase. Commit sa robí po úpravách kódu, ktoré vyjadrujú malú zmenu functionality
- **Branch** - iná verzia kódu, na ktorej sa dá pracovať bez toho, aby sa upravovala hlavná verzia, rôzne verzie (branches) sa dajú neskôr zjednotiť cez merge alebo pull request
- **Push** - aktualizácie kódu na strane Githubu mojou verziou
- **Pull** - aktualizácie na mojej strane najnovšiou verziou na Githube
- **Pull request** - slúži na mergovanie 2 branches. V rámci pull requestu sa vykonáva code review a diskutuje sa o zmanách kde sa rozhodne či sa vykoná merge

## Základný workflow

1. Vytvorenie issue (user task), estimovanie a priradenie
2. Programátor si vytvorí nový branch s názvom for formáte "i[číslo issue]"
3. Programátor následne pracuje na tejto issue.
4. Programátor vytvorí pull request
5. Následne reviewer vykoná core review
6. Podľa výsledkov z code review sa vykoná merge alebo sa požiadá o ďalšie zmeny

## Code review

Čo sledovať, resp. čo by mal kód spĺňať:

- Kód by sa mal riadiť definovanými metodikami (názvy premien, odsadzovanie, atď..)
- Žiadna duplicita kódu
- Funkčné testy
- Kód musí byť skompilovateľný bez chýb
  - K tomuto pomáha nástroj TravisCI, ktorý automaticky spustí build pre každý pull request a povie, či bol úspešný

# Obsah Github Wiki

## Consult some git tutorials:

- Very simple: <http://rogerdudler.github.io/git-guide/>
- Interactive: <https://learngitbranching.js.org/> or <https://try.github.io/levels/1/challenges/1> or <http://git.rocks/getting-started/>
- Official doc: <https://git-scm.com/docs>
- Gui for windows (if you dont like command line): <https://git-scm.com/download/gui/windows>
- Some IDEs have integrated support for git, such as IntelliJ Idea, Eclipse, even [Atom.io](http://Atom.io)

---

Each issue should be worked on in a separate branches called "i[number of issue]".

Example: fixing issue #21 would result in creating a new branch called i21

## Create new branch:

```
git checkout -b i21
```

This would also move all uncommitted changes from the branch you are currently at.

```
# git status
```

On branch master

Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean

```
# git checkout -b test
```

Switched to a new branch 'test'

... DO YOUR WORK - COMMIT CHANGES ...

(as many as you like)

## Example commit:

```
# git add .
```

```
# git commit -m "change font color for h1"
```

## Example Push:

```
# git push
```

## rollback a github repository to a specific commit

```
# git reset --hard <old-commit-id>
```

```
# git push -f <remote-name> <branch-name>
```

When you think issue is ready, go to GitHub site and create a pull request, place the issue into Review/QA pipeline

# Metodika testovania

Bc. Marián HeseK

Vedúci projektu: Ing. Rastislav Bencel  
Názov tímu: MGWA - Make Wifi Great Again  
Členovia tímu: Bc. Katarína Bedejová  
Bc. Matej Belluš  
Bc. Marcel Sabol  
Bc. Michal Gottstein  
Bc. Marián HeseK  
Bc. Michal Hampel  
Bc. Miroslav Procházka  
Pracovisko: FIIT STU - 5.45  
Akademický rok: 2017/2018  
Posledná zmena: 11.5.2018

# Obsah

<b>1 Úvod</b>	<b>3</b>
<b>2 Floodlight - Unit Tests</b>	<b>3</b>
2.1 Písanie nového testu	4
2.2 Príklad JUnit Testu	4
<b>3 Open vSwitch - Unit Tests</b>	<b>5</b>

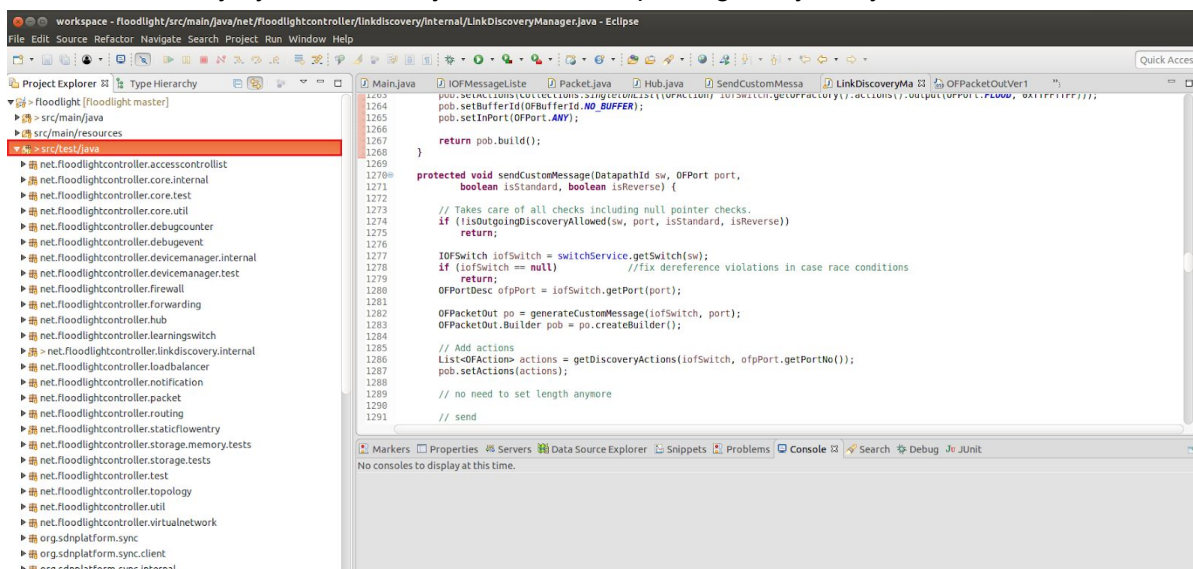
# 1 Úvod

V našom projekte využívame tieto technológie: SDN kontrolér Floodlight a viacvrstvový virtuálny prepínač Open vSwitch. Obi dva tieto opensource projekty obsahujú vlastné unit testy, ktorými sa dá otestovať ich celková funkcionálnosť a takisto vieme vytvoriť vlastné testy. Keďže tieto technológie budeme editovať a dopĺňať novú funkcionálnosť, unit testami si vieme otestovať, či sme nepokazili ich pôvodnú funkčnosť a otestovať tiež naše nové funkcie.

## 2 Floodlight - Unit Tests

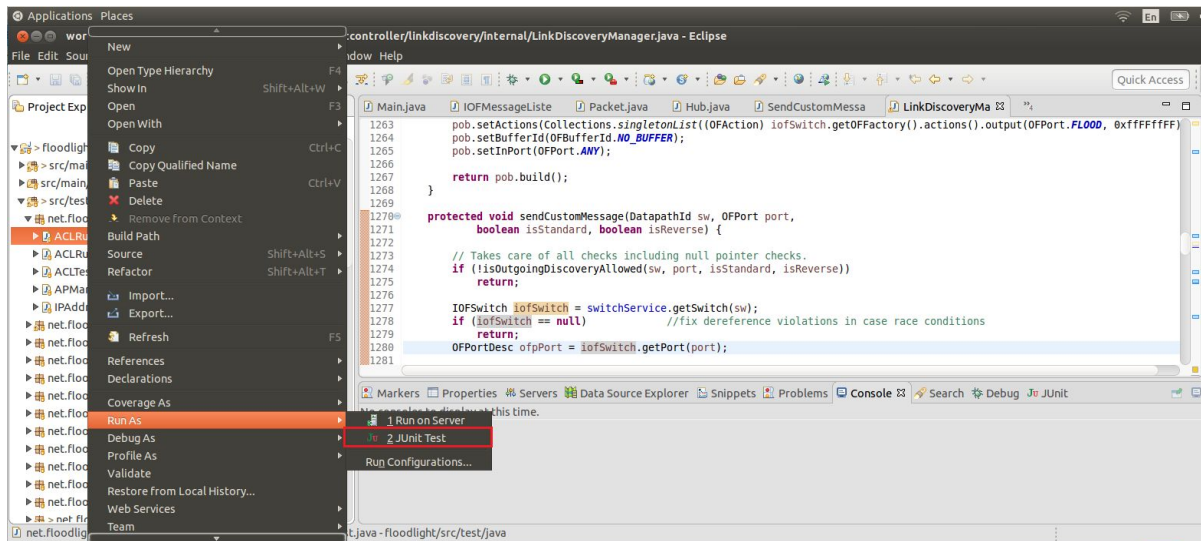
Po importovaní zdrojových kódov do nami zvoleného IDE Eclipse Oxygen, vieme jednoducho spustiť jednotlivé unit testy. Postup je popísaný na nasledujúcich obrázkoch.

- Testy sú vytvorené samostatne v `/src/test/java`. Budeme sa držať tejto konvencie a nové testy vytvárať tu vždy v konkrétnom package a výstižným názvom.



Obr. 1: Cesta k unit testom

- Konkrétny test spustíme kliknutím pravého myšidla na test -> Run as... -> JUnit Test



Obr.2: Run JUnit Test

## 2.1 Písanie nového testu

Pri písaní nového testu budeme dodržiavať "AAA (Arrange, Act, Assert)" vzor. Tento vzor je zložený z 3 častí:

1. **Arrange** - nastavujú sa tu hodnoty dát a inicializujú objekty
2. **Act** - volajú sa tu testovacie metódy s dátami zo sekcie Arrange
3. **Assert** - prebieha tu verifikácia, či testovací scenár prebehol správne

Testy sú v zdrojovom kóde anotované `@Test`.

## 2.2 Príklad JUnit Testu

```
public class IPAddressUtilTest extends FloodlightTestCase {
    @Test
    public void testIsSubnet(){
        assertFalse(IPAddressUtil.isSubnet("10.0.0.1/32", "10.0.0.2/32"));
        assertTrue(IPAddressUtil.isSubnet("10.0.0.1/8", "10.0.0.2/8"));
        assertTrue(IPAddressUtil.isSubnet("10.0.0.1/32", "10.0.0.2/8"));
        assertFalse(IPAddressUtil.isSubnet("10.0.0.1/8", "10.0.0.2/32"));
        assertTrue(IPAddressUtil.isSubnet("10.0.0.1/8", null));
        assertFalse(IPAddressUtil.isSubnet(null, "10.0.0.2/32"));
    }
}
```

## 3 Open vSwitch - Unit Tests

Unit testy pre OVS sú uložené v ../ovs/tests. Nové testy budeme tiež vytvárať na toto miesto. Spustenie unit testov pre Open vSwitch zabezpečíme spustením nasledovných príkazov:

1. *make check* - ak chceme spustiť všetky unit testy za sebou
2. *make check TESTSUITEFLAGS=-j8* - ak chceme spustiť testy paralelne, vytvoríme 8 vlákien
3. *make check TESTSUITEFLAGS=--list* - ak chceme zobrazíť zoznam dostupných testov
4. *make check TESTSUITEFLAGS='123 477-484'* - ak chceme spustiť konkrétne testy
5. *make check TESTSUITEFLAGS=--help* - ak chceme zobrazíť všetky testovacie možnosti

# Metodika úloh

Bc. Marcel Sabol

Vedúci projektu: Ing. Rastislav Bencel  
Názov tímu: MGWA - Make Wifi Great Again  
Členovia tímu: Bc. Katarina Bedejová  
Bc. Matej Belluš  
Bc. Marcel Sabol  
Bc. Michal Gottstein  
Bc. Marián Hesek  
Bc. Michal Hampel  
Bc. Miroslav Procházka  
Pracovisko: FIIT STU - 5.45  
Akademický rok: 2017/2018  
Dátum odovzdania: 11.5.2018



# Obsah

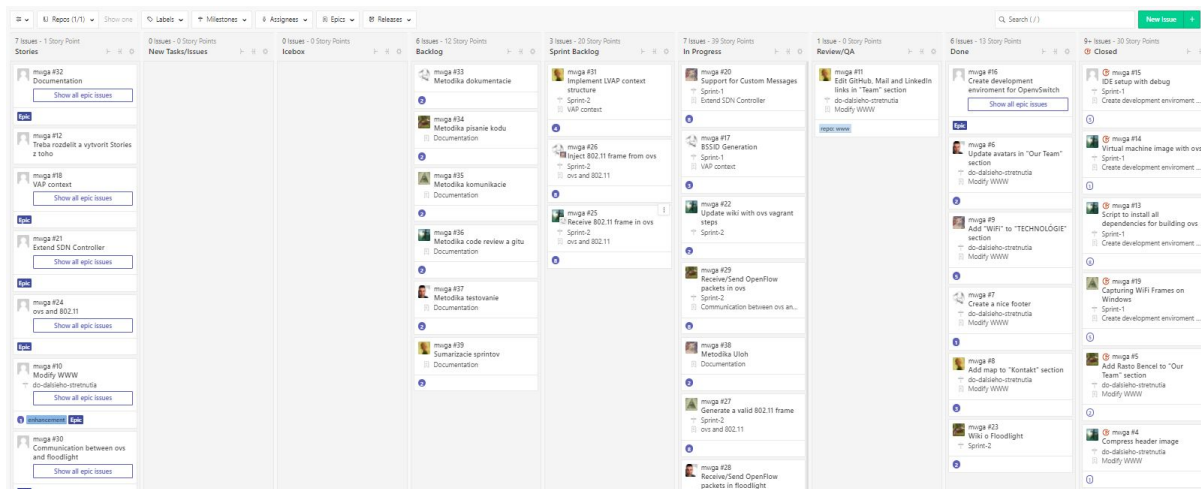
<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Spravovanie úloh v ZenHube</b>	<b>4</b>
2.1	Vytvorenie úlohy	4
2.2	Vymazanie úlohy	7
2.3	Pridelenie a estimovanie úlohy	7
2.3	Stavy úlohy	7

# 1 Úvod

Táto metodika definuje postúpi a pravidla pri práci s projektovými úlohami (taskami). Na vytváranie, editovanie a prideľovanie všetkých projektových úloh je použitý nástroj ZenHub ([www.zenhub.com](http://www.zenhub.com)). Tento nástroj bol vybraný z dôvodov, že je online dostupný, predstavuje nadstavbu nad GitHubom a má jednoduché ovládanie.

## 2 Spravovanie úloh v ZenHube

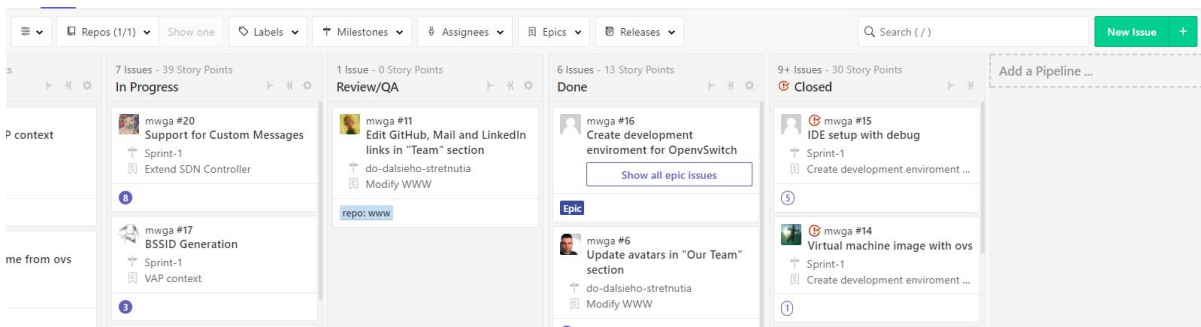
Táto kapitola dokumentuje postúpi pri práci s projektovými úlohami v ZenHube. V ZenHube sa úloha nazýva issue. Issue môže reprezentovať story alebo úlohu, preto je potrebné sa zamyslieť o aký typ úlohy ide. Epica predstavuje najvyššiu abstrakciu a v našom projekte nie je tato abstrakcia použitá.




Dôvod je ten, že GitHub nepodporuje „vnorenú issue“ teda vloženie issue do issue. Z tohto dôvodu na manažovanie taskou používame 2 abstrakcie. Issue typu epic reprezentuje v našom projekte story task. Toto riešenie sme zvolili kvôli potrebe vedieť vytvárať story tasky viac ako vedieť vytvárať epici. Tento story task reprezentuje väčšiu funkcionality (napr. samostatný modul). Úloha pokrýva malú samostatnú funkcionality, ktorá dokáže byť samostatne otestovaná a jej implementačná náročnosť nie je vysoká.

### 2.1 Vytvorenie úlohy

Ako bolo vyššie spomenuté, vytvorenie úlohy v ZenHube je v podstate vytvorenie issue. (pravý horný roh obrazovky)



MWGA/mwga nova uloha

xsabolm

Issue title

nova uloha

Write Preview

Opis ulohy

Attach files by selecting them here or pasting from the clipboard.

Styling with Markdown is supported

Create an Epic Submit new Issue

Pipeline

Stories

Labels

No Labels yet

Assignees

No one - assign yourself

Milestone

No Milestone


Estimate

No estimate yet

Epics

Pri vytváraní úlohy, sa nastaví názov úlohy a jej popis (čo, ako treba implementovať, zmeniť, vymazať). Ako bolo spomenuté vyššie, ak tvorca tasky vyberie možnosť "Create an Epic" pôjde o story tasku. Ak zvolí "Submit new Issue" ide o obyčajnú tasku.

MWGA/mwga#32 Documentation

Open

Documentation #32

create documentation for Floodlight, OPVS

Documentation has no dependencies

+ add dependency

Documentation is an Epic

0 of 6 Issues completed

0 of 12 Epic Points completed

Backlog

mwga#34 Metodika pisanie kodu

mwga#36 Metodika code review a gitu

mwga#37 Metodika testovanie

mwga#39 Sumarizacie sprintov

In Progress

mwga#35 Metodika komunikacie

Pipeline

Stories

Labels

Epic

Assignees

No one - assign yourself

Milestone

No Milestone

Estimate

12 epic points

Obrázok zobrazuje príklad story tasku. Rôzne úlohy, ktoré sú súčasťou tohto story tasku sú pridávané pod daný story task.

MWGA/mwga#34 Metodika pisanie kodu

**Metodika pisanie kodu #34**

- ako pisat funkciec
- nazvy premennych
- odsadzovanie, ...

Metodika pisanie kodu has no dependencies [+ add dependency](#)

[+ See 1 older event](#)

- Horkyze added this issue to [MWGA/mwga#32 Documentation](#) 13 hours ago
- Horkyze changed the pipeline from **Stories** to **Backlog** 13 hours ago
- Horkyze set the estimate to **2** 13 hours ago

xsabolm

[Write](#) [Preview](#)

Write a comment...

**Pipeline**

Backlog

**Labels**

No Labels yet

**Assignees**

MisoHampel

**Milestone**

No Milestone

**Estimate**

**2**

**Labels**

No Labels yet

**Assignees**

MisoHampel

**Milestone**

No Milestone

**Estimate**

**2**

**Epics**

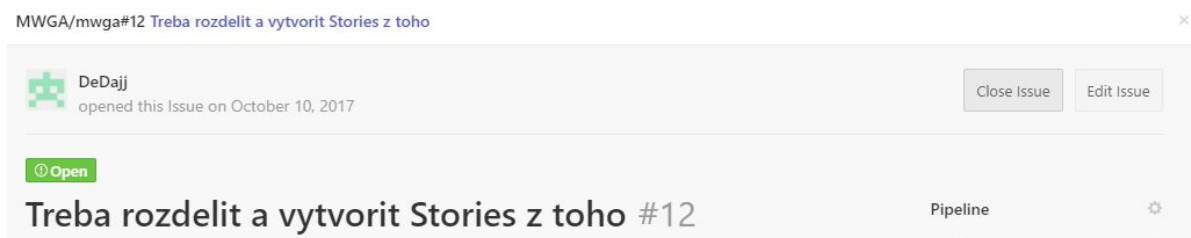
Documentation

Tieto dve obrázky zobrazujú obyčajnú tasku. V zobrazených detailov tasky je možno vidieť zmeny v stavov úlohy.

- pridanie úlohy pod story task Documentation
- presunutie tasku do iného stavu (životný cyklus tasky)
- estimovanie úlohy

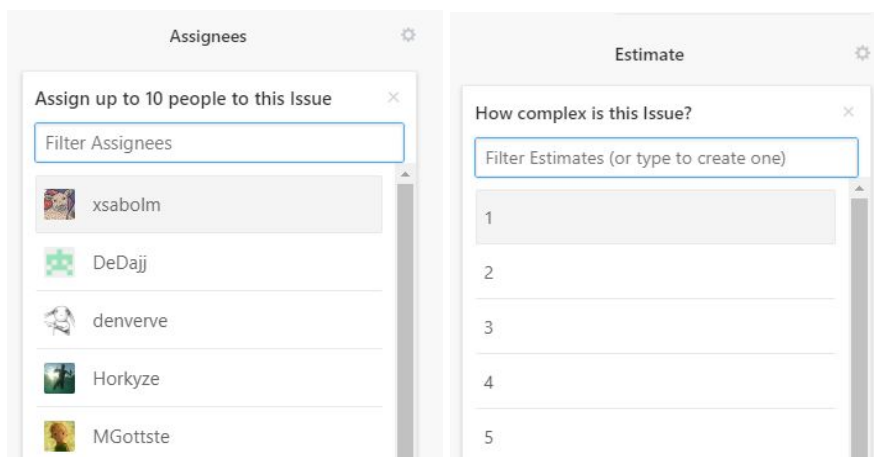
## 2.2 Vymazanie úlohy

Vymazanie tasku musí byť pre konzultované na stretnutí celého tímu. Ide zväčša o zistenie, že daná taska nemá význam z hľadiska požiadaviek a funkcionality projektu. ZenHub nemá implementovanú funkcionality na vymazanie tasky, takže naše riešenie je uzavrieť tasku s komentom, ktorý vysvetľuje, prečo bola daná taska predčasne uzavretá. Na zatvorenie tasky slúži tlačidlo “close issue”.



## 2.3 Pridelenie a estimovanie úlohy

Pridelenie a estimovanie úloh sa nastavuje v detailov úlohy. Tasky sú pridelené členom tímu po dohode pri stretnutiach alebo každý člen tímu si sám sebe prideli úlohu akú chce.



Úlohy sú estimované pri ich vytvorení alebo na najbližšom stretnutí tímu ak bola taska vytvorená mimo stretnutí (neobvyklý ale možný scenár). Na estimovanie úloh používame nástroj <https://www.pointingpoker.com/>.

## 2.3 Stavy úloh

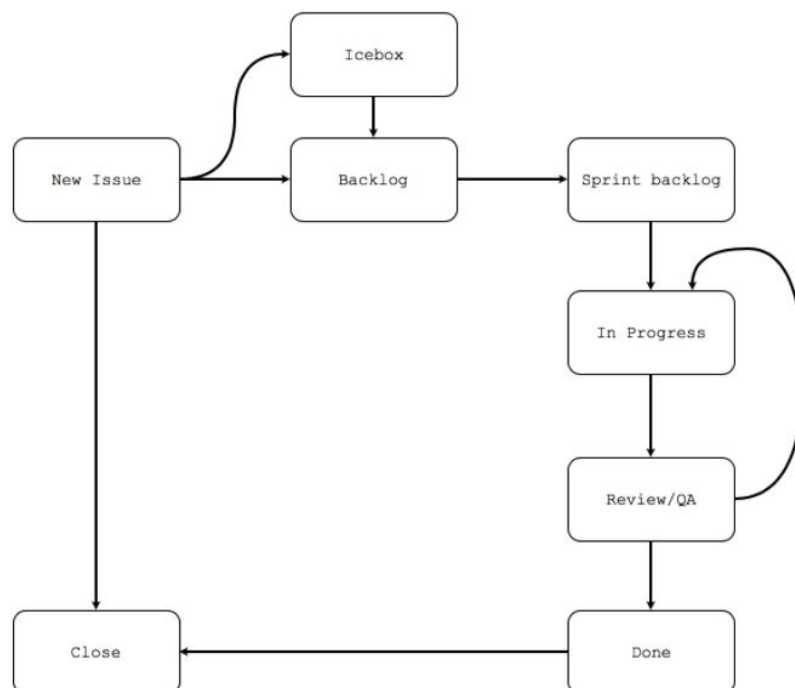
Každá vytvorená taska ma počas svojho životného cyklu isté stavy. V tejto podkapitole sú jednotlivé stavy popísané.



- Stories - nachádzajú sa tu story tasky. Story taska je rozdrobená na menšie tasky. Story taska sa uzavrie až keď sú všetky menšie tasky implementované a uzavreté.
- New Task/Issues - v tomto stave sú novo neestimované vytvorené úlohy.
- IceBox - v tomto stave sa nachádzajú tasky, ktoré sú síce oestimované ale nedajú sa implementovať pretože nadväzujú na ešte neimplementované tasky
- Backlog - v tomto stave sú oestimované tasky, ktoré sa môžu v nasledujúcom šprinte implementovať

3 Issues - 20 Story Points Sprint Backlog	8 Issues - 41 Story Points In Progress	1 Issue - 0 Story Points Review/QA	6 Issues - 13 Story Points Done	9+ Issues - 30 Story Points Closed
----------------------------------------------	-------------------------------------------	---------------------------------------	------------------------------------	---------------------------------------

- Sprint Backlog - v Sprint backlogu sa nachádzajú ešte nepridelené tasky, ktoré sa budú počas sprintu implementovať
- In Progress - v tomto stave sú tasky, ktoré boli v sprint backlogu a boli pridelené členovi tímu
- Review/QA - po dokončení implementácie je taska poslaná na kontrolu inému členovi alebo členom tímu, ktorý zhodnotia implementáciu (skontrolujú kód a otestujú implementovanú funkcionálnosť)
- Done - ak bola taska schválená v časti review tak je v Done stave až do konca sprintu.
- Closed - na konci sprintu sa každá taska v review zhodnotí, uzavrie a uloží do stavu closed.



## 2.4 Definition of Done

Definovanie, ktorá úloha je "done". Úloha je presnutá do stavu "done",

V prípade, že:

1. Osoba, ktorá pracovala na danej úlohe, spravil všetko čo bolo cieľom úlohy
2. Ak išlo o funkcionality na, ktorú sa dá vytvoriť test, tak vytvoril test pre danú úlohu.
3. Spustil všetky zatiaľ existujúce testy, a tie sa ukončili úspešne
4. Člen tímu, ktorý pracoval na úlohe, pridá do dokumentácie na wiki v repozitáre mwga dokumentáciu opisujúcu čo zmenil a s ukážkami z kódu opíše workflow jeho úlohy.
5. Úlohu dá z "In progres" stavu do stavu "review"
6. Úlohu reviewne iný člen tímu, ktorý spustí testy a skontroluje kód úlohy a pridanie dokumentácie.
7. Po absolvovaní všetkých krokov je úloha považovaná za ukončenú a daná do stĺpca done





